Contents

What's New in SAS/ACCESS 9.4 for Relational Databases ...................... xi
Accessibility ................................................................. xxiii

PART 1  Concepts  1

Chapter 1 • Overview of SAS/ACCESS Interface to Relational Databases .......... 3
  About This Document ...................................................... 3
  Methods for Accessing Relational Database Data .......................... 4
  Selecting a SAS/ACCESS Method .......................................... 4
  SAS Views of DBMS Data .................................................. 6
  Choosing Your Degree of Numeric Precision ................................ 7

Chapter 2 • SAS Names and Support for DBMS Names .......................... 13
  DBMS-Specific Naming Conventions ...................................... 13
  SAS Naming Conventions ................................................... 14
  SAS/ACCESS Default Naming Behaviors .................................... 15
  Renaming DBMS Data ....................................................... 16
  Options That Affect SAS/ACCESS Naming Behavior ....................... 17
  Naming Behavior When Retrieving DBMS Data ............................ 17
  Naming Behavior When Creating DBMS Objects ........................... 18
  SAS/ACCESS Naming Examples ............................................ 19

Chapter 3 • Data Integrity and Security ....................................... 27
  Introduction to Data Integrity and Security ............................... 27
  DBMS Security ............................................................. 27
  SAS Security ............................................................... 28
  Potential Result Set Differences When Processing Null Data ............ 33

Chapter 4 • Performance Considerations ..................................... 37
  Increasing Throughput of the SAS Server ................................... 37
  Limiting Retrieval .......................................................... 37
  Repeatedly Accessing Data ................................................ 39
  Sorting DBMS Data .......................................................... 39
  Temporary Table Support for SAS/ACCESS .................................. 40

Chapter 5 • Optimizing Your SQL Usage ..................................... 43
  Overview: Optimizing Your SQL Usage .................................... 43
  Passing Functions to the DBMS Using PROC SQL ......................... 44
  Passing Joins to the DBMS .................................................. 45
  Passing the DELETE Statement to Empty a Table ......................... 46
  When Passing Joins to the DBMS Will Fail ................................ 47
  Passing DISTINCT and UNION Processing to the DBMS .................. 48
  Optimizing the Passing of WHEREClauses to the DBMS ................... 49
  Using the DBINDEX=, DBKEY=, and MULTI_DATASRC_OPT= Options ...... 50

Chapter 6 • Threaded Reads ................................................... 53
  Overview: Threaded Reads in SAS/ACCESS ............................... 53
<table>
<thead>
<tr>
<th>Chapter 7 • National Language Support</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: NLS for SAS/ACCESS</td>
<td>63</td>
</tr>
<tr>
<td>Options</td>
<td>63</td>
</tr>
<tr>
<td>Data Types</td>
<td>63-64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8 • How SAS/ACCESS Works</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to How SAS/ACCESS Works</td>
<td>65</td>
</tr>
<tr>
<td>How the SAS/ACCESS LIBNAME Statement Works</td>
<td>66</td>
</tr>
<tr>
<td>How the SQL Pass-Through Facility Works</td>
<td>67</td>
</tr>
<tr>
<td>How the ACCESS Procedure Works</td>
<td>68</td>
</tr>
<tr>
<td>How the DBLOAD Procedure Works</td>
<td>69</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 9 • In-Database Processing with SAS/ACCESS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: In-Database Processing</td>
<td>71</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PART 2 • General Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 10 • SAS/ACCESS Features by Host</td>
<td>77</td>
</tr>
<tr>
<td>Introduction</td>
<td>77-78</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Aster: Supported Features</td>
<td>78</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to DB2 under UNIX and PC Hosts: Supported Features</td>
<td>78</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to DB2 under z/OS: Supported Features</td>
<td>79</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Greenplum: Supported Features</td>
<td>79</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Hadoop: Supported Features</td>
<td>80</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to HAWQ: Supported Features</td>
<td>81</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Impala: Supported Features</td>
<td>82</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Informix: Supported Features</td>
<td>82</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Microsoft SQL Server: Supported Features</td>
<td>83</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to MySQL: Supported Features</td>
<td>83</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Netezza: Supported Features</td>
<td>84</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to ODBC: Supported Features</td>
<td>85</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to OLE DB: Supported Features</td>
<td>86</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Oracle: Supported Features</td>
<td>86</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to PostgreSQL: Supported Features</td>
<td>87</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to SAP HANA: Supported Features</td>
<td>88</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Sybase: Supported Features</td>
<td>88</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Sybase IQ: Supported Features</td>
<td>89</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Teradata: Supported Features</td>
<td>90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 11 • The LIBNAME Statement for Relational Databases</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: LIBNAME Statement for Relational Databases</td>
<td>93-96</td>
</tr>
</tbody>
</table>
Assigning a Libref Interactively .................................................. 97
Dictionary ................................................................. 97

**Chapter 12 • Data Set Options for Relational Databases** ........................................ 237
About the Data Set Options for Relational Databases ........................................ 241
Dictionary ................................................................. 242

**Chapter 13 • Macro Variables and System Options for Relational Databases** ........ 445
Introduction to Macro Variables and System Options ..................................... 445
Macro Variables for Relational Databases ................................................ 445
System Options for Relational Databases .................................................. 447
Dictionary ................................................................. 448

**Chapter 14 • SQL Pass-Through Facility for Relational Databases** ..................... 471
About SQL Procedure Interactions ................................................... 471
Syntax: SQL Pass-Through Facility for Relational Databases ....................... 472
Dictionary ................................................................. 473

**PART 3  DBMS-Specific Reference**  483

**Chapter 15 • SAS/ACCESS Interface to Aster** ............................................. 485
Introduction to SAS/ACCESS Interface to Aster ......................................... 486
LIBNAME Statement Specifics for Aster ................................................ 486
Data Set Options for Aster .......................................................... 490
SQL Pass-Through Facility Specifics for Aster ......................................... 492
Autopartitioning Scheme for Aster ....................................................... 494
Passing SAS Functions to Aster ........................................................ 495
Passing Joins to Aster ........................................................................ 496
Bulk Loading and Unloading for Aster ................................................... 497
Naming Conventions for Aster ............................................................. 499
Data Types for Aster ....................................................................... 499

**Chapter 16 • SAS/ACCESS Interface to DB2 for UNIX and PC Hosts** ................. 503
Introduction to SAS/ACCESS Interface to DB2 under UNIX and PC Hosts ........ 504
LIBNAME Statement Specifics for DB2 under UNIX and PC Hosts ................. 504
Data Set Options for DB2 under UNIX and PC Hosts ................................ 508
SQL Pass-Through Facility Specifics for DB2 under UNIX and PC Hosts .......... 511
Autopartitioning Scheme for DB2 under UNIX and PC Hosts ..................... 513
Temporary Table Support for DB2 under UNIX and PC Hosts ...................... 516
Calling Stored Procedures in DB2 under UNIX and PC Hosts ....................... 516
DBLOAD Procedure Specifics for DB2 under UNIX and PC Hosts .................. 518
Passing SAS Functions to DB2 under UNIX and PC Hosts ........................... 520
Passing Joins to DB2 under UNIX and PC Hosts ...................................... 521
Bulk Loading for DB2 under UNIX and PC Hosts ....................................... 521
Locking in the DB2 under UNIX and PC Hosts Interface ............................ 525
Naming Conventions for DB2 under UNIX and PC Hosts ............................. 526
Data Types for DB2 under UNIX and PC Hosts .......................................... 527

**Chapter 17 • SAS/ACCESS Interface to DB2 for z/OS** ..................................... 531
Introduction to SAS/ACCESS Interface to DB2 under z/OS .......................... 533
LIBNAME Statement Specifics for DB2 under z/OS .................................... 533
Data Set Options for DB2 under z/OS ................................................... 536
SQL Pass-Through Facility Specifics for DB2 under z/OS ............................. 539
Autopartitioning Scheme for DB2 under z/OS .......................................... 540
<table>
<thead>
<tr>
<th>Chapter 18 • SAS/ACCESS Interface to Greenplum</th>
<th>583</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to Greenplum</td>
<td>584</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for Greenplum</td>
<td>584</td>
</tr>
<tr>
<td>Data Set Options for Greenplum</td>
<td>588</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for Greenplum</td>
<td>590</td>
</tr>
<tr>
<td>Autopartitioning Scheme for Greenplum</td>
<td>592</td>
</tr>
<tr>
<td>Passing SAS Functions to Greenplum</td>
<td>594</td>
</tr>
<tr>
<td>Passing Joins to Greenplum</td>
<td>595</td>
</tr>
<tr>
<td>Bulk Loading for Greenplum</td>
<td>595</td>
</tr>
<tr>
<td>Naming Conventions for Greenplum</td>
<td>599</td>
</tr>
<tr>
<td>Data Types for Greenplum</td>
<td>599</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 19 • SAS/ACCESS Interface to Hadoop</th>
<th>603</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to Hadoop</td>
<td>604</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for Hadoop</td>
<td>605</td>
</tr>
<tr>
<td>Data Set Options for Hadoop</td>
<td>610</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for Hadoop</td>
<td>611</td>
</tr>
<tr>
<td>Passing SAS Functions to Hadoop</td>
<td>612</td>
</tr>
<tr>
<td>Passing Joins to Hadoop</td>
<td>613</td>
</tr>
<tr>
<td>Bulk Loading for Hadoop</td>
<td>613</td>
</tr>
<tr>
<td>Naming Conventions for SAS and Hive</td>
<td>615</td>
</tr>
<tr>
<td>Data Types for Hadoop</td>
<td>615</td>
</tr>
<tr>
<td>Sample Code for Hadoop</td>
<td>626</td>
</tr>
<tr>
<td>Using SAS In-Database Processing with Hadoop</td>
<td>627</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 20 • SAS/ACCESS Interface to HAWQ</th>
<th>629</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to HAWQ</td>
<td>630</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for HAWQ</td>
<td>630</td>
</tr>
<tr>
<td>Data Set Options for HAWQ</td>
<td>634</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for HAWQ</td>
<td>636</td>
</tr>
<tr>
<td>Autopartitioning Scheme for HAWQ</td>
<td>638</td>
</tr>
<tr>
<td>Passing SAS Functions to HAWQ</td>
<td>640</td>
</tr>
<tr>
<td>Passing Joins to HAWQ</td>
<td>640</td>
</tr>
<tr>
<td>Bulk Loading for HAWQ</td>
<td>641</td>
</tr>
<tr>
<td>Naming Conventions for HAWQ</td>
<td>644</td>
</tr>
<tr>
<td>Data Types for HAWQ</td>
<td>645</td>
</tr>
</tbody>
</table>
Chapter 21 • SAS/ACCESS Interface to Impala .................................................. 649
  Introduction to SAS/ACCESS Interface to Impala ........................................ 650
  LIBNAME Statement Specifics for Impala ..................................................... 650
  Data Set Options for Impala ........................................................................... 653
  SQL Pass-Through Facility Specifics for Impala ............................................. 654
  Passing SAS Functions to Impala ................................................................. 655
  Passing Joins to Impala .................................................................................. 656
  Bulk Loading for Impala ................................................................................ 656
  Naming Conventions for Impala ..................................................................... 658
  Data Types for Impala .................................................................................... 659

Chapter 22 • SAS/ACCESS Interface to Informix ............................................... 663
  Introduction to SAS/ACCESS Interface to Informix ....................................... 664
  LIBNAME Statement Specifics for Informix ................................................... 664
  Data Set Options for Informix ........................................................................ 667
  SQL Pass-Through Facility Specifics for Informix ......................................... 668
  Autopartitioning Scheme for Informix ............................................................ 671
  Temporary Table Support for Informix ........................................................... 672
  Passing SAS Functions to Informix ............................................................... 672
  Passing Joins to Informix ............................................................................... 673
  Locking in the Informix Interface .................................................................. 673
  Naming Conventions for Informix ................................................................. 674
  Data Types for Informix ................................................................................ 674
  Informix Servers ............................................................................................ 677

Chapter 23 • SAS/ACCESS Interface to Microsoft SQL Server .......................... 681
  Introduction to SAS/ACCESS Interface to Microsoft SQL Server ................. 681
  LIBNAME Statement Specifics for Microsoft SQL Server .............................. 682
  Data Set Options for Microsoft SQL Server ................................................. 686
  SQL Pass-Through Facility Specifics for Microsoft SQL Server ..................... 688
  DBLOAD Procedure Specifics for Microsoft SQL Server .............................. 690
  Passing SAS Functions to Microsoft SQL Server ......................................... 691
  Locking in the Microsoft SQL Server Interface .......................................... 691
  Naming Conventions for Microsoft SQL Server .......................................... 693
  Data Types for Microsoft SQL Server .......................................................... 693

Chapter 24 • SAS/ACCESS Interface to MySQL ............................................... 697
  Introduction to SAS/ACCESS Interface to MySQL ........................................ 697
  LIBNAME Statement Specifics for MySQL .................................................... 698
  Data Set Options for MySQL .......................................................................... 700
  SQL Pass-Through Facility Specifics for MySQL ......................................... 702
  Autocommit and Table Types ....................................................................... 703
  Understanding MySQL Update and Delete Rules ....................................... 703
  Passing SAS Functions to MySQL ............................................................... 704
  Passing Joins to MySQL ............................................................................... 705
  Bulk Loading for MySQL .............................................................................. 705
  Naming Conventions for MySQL ................................................................. 706
  Case Sensitivity for MySQL .......................................................................... 707
  Data Types for MySQL .................................................................................. 707

Chapter 25 • SAS/ACCESS Interface to Netezza ............................................... 711
  Introduction to SAS/ACCESS Interface to Netezza ....................................... 711
  LIBNAME Statement Specifics for Netezza .................................................... 712
  Data Set Options for Netezza ......................................................................... 716
  SQL Pass-Through Facility Specifics for Netezza ......................................... 718
  Temporary Table Support for Netezza ............................................................ 719
<table>
<thead>
<tr>
<th>Chapter 26 • SAS/ACCESS Interface to ODBC</th>
<th>727</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to ODBC</td>
<td>728</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for ODBC</td>
<td>732</td>
</tr>
<tr>
<td>Data Set Options for ODBC</td>
<td>737</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for ODBC</td>
<td>738</td>
</tr>
<tr>
<td>Autopartitioning Scheme for ODBC</td>
<td>743</td>
</tr>
<tr>
<td>DBLOAD Procedure Specifics for ODBC</td>
<td>747</td>
</tr>
<tr>
<td>Temporary Table Support for ODBC</td>
<td>749</td>
</tr>
<tr>
<td>Passing SAS Functions to ODBC</td>
<td>749</td>
</tr>
<tr>
<td>Passing Joins to ODBC</td>
<td>750</td>
</tr>
<tr>
<td>Bulk Loading for ODBC</td>
<td>750</td>
</tr>
<tr>
<td>Locking in the ODBC Interface</td>
<td>750</td>
</tr>
<tr>
<td>Naming Conventions for ODBC</td>
<td>752</td>
</tr>
<tr>
<td>Data Types for ODBC</td>
<td>752</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 27 • SAS/ACCESS Interface to OLE DB</th>
<th>755</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to OLE DB</td>
<td>755</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for OLE DB</td>
<td>756</td>
</tr>
<tr>
<td>Data Set Options for OLE DB</td>
<td>762</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for OLE DB</td>
<td>764</td>
</tr>
<tr>
<td>Temporary Table Support for OLE DB</td>
<td>769</td>
</tr>
<tr>
<td>Passing SAS Functions to OLE DB</td>
<td>769</td>
</tr>
<tr>
<td>Passing Joins to OLE DB</td>
<td>770</td>
</tr>
<tr>
<td>Bulk Loading for OLE DB</td>
<td>771</td>
</tr>
<tr>
<td>Locking in the OLE DB Interface</td>
<td>771</td>
</tr>
<tr>
<td>Accessing OLE DB for OLAP Data</td>
<td>772</td>
</tr>
<tr>
<td>Naming Conventions for OLE DB</td>
<td>775</td>
</tr>
<tr>
<td>Data Types for OLE DB</td>
<td>775</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 28 • SAS/ACCESS Interface to Oracle</th>
<th>779</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to Oracle</td>
<td>780</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for Oracle</td>
<td>780</td>
</tr>
<tr>
<td>Data Set Options for Oracle</td>
<td>784</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for Oracle</td>
<td>787</td>
</tr>
<tr>
<td>Autopartitioning Scheme for Oracle</td>
<td>789</td>
</tr>
<tr>
<td>ACCESS Procedure Specifics for Oracle</td>
<td>792</td>
</tr>
<tr>
<td>DBLOAD Procedure Specifics for Oracle</td>
<td>794</td>
</tr>
<tr>
<td>Maximizing Oracle Performance</td>
<td>796</td>
</tr>
<tr>
<td>Passing SAS Functions to Oracle</td>
<td>796</td>
</tr>
<tr>
<td>Passing Joins to Oracle</td>
<td>797</td>
</tr>
<tr>
<td>Bulk Loading for Oracle</td>
<td>798</td>
</tr>
<tr>
<td>Locking in the Oracle Interface</td>
<td>800</td>
</tr>
<tr>
<td>Naming Conventions for Oracle</td>
<td>801</td>
</tr>
<tr>
<td>Data Types for Oracle</td>
<td>802</td>
</tr>
<tr>
<td>Examples</td>
<td>795</td>
</tr>
<tr>
<td>Examples</td>
<td>793</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 29 • SAS/ACCESS Interface to PostgreSQL</th>
<th>811</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to PostgreSQL</td>
<td>811</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for PostgreSQL</td>
<td>812</td>
</tr>
</tbody>
</table>
Contents

Chapter 30 • SAS/ACCESS Interface to SAP HANA ........................................... 827
  Introduction to SAS/ACCESS Interface to SAP HANA ................................. 828
  LIBNAME Statement Specifics for SAP HANA ........................................... 828
  Data Set Options for SAP HANA .......................................................... 834
  SQL Pass-Through Facility Specifics for SAP HANA .................................. 836
  Understanding SAP HANA Update and Delete Rules ................................ 837
  Autopartitioning Scheme for SAP HANA ............................................... 838
  SAP HANA Schema Flexibility ................................................................ 839
  Passing SAS Functions to SAP HANA ..................................................... 839
  Passing Joins to SAP HANA .................................................................. 840
  Bulk Loading for SAP HANA .................................................................. 841
  Naming Conventions for SAP HANA ....................................................... 841
  Data Types for SAP HANA .................................................................... 842

Chapter 31 • SAS/ACCESS Interface to Sybase ............................................... 849
  Introduction to SAS/ACCESS Interface to Sybase .................................... 850
  LIBNAME Statement Specifics for Sybase ............................................... 850
  Data Set Options for Sybase .................................................................. 854
  SQL Pass-Through Facility Specifics for Sybase ....................................... 855
  Autopartitioning Scheme for Sybase ..................................................... 856
  Temporary Table Support for Sybase ..................................................... 858
  ACCESS Procedure Specifics for Sybase ............................................... 858
  DBLOAD Procedure Specifics for Sybase ............................................... 859
  Passing SAS Functions to Sybase .......................................................... 861
  Passing Joins to Sybase ........................................................................ 862
  Bulk Loading for Sybase ....................................................................... 862
  Reading Multiple Sybase Tables ............................................................ 862
  Locking in the Sybase Interface ............................................................. 863
  Naming Conventions for Sybase ............................................................ 864
  Case Sensitivity in Sybase ..................................................................... 865
  Data Types for Sybase ........................................................................... 866
  National Language Support for Sybase ................................................... 871

Chapter 32 • SAS/ACCESS Interface to Sybase IQ ......................................... 873
  Introduction to SAS/ACCESS Interface to Sybase IQ ............................... 874
  LIBNAME Statement Specifics for Sybase IQ .......................................... 874
  Data Set Options for Sybase IQ .............................................................. 878
  SQL Pass-Through Facility Specifics for Sybase IQ .................................. 880
  Autopartitioning Scheme for Sybase IQ .................................................. 881
  Passing SAS Functions to Sybase IQ ...................................................... 883
  Passing Joins to Sybase IQ .................................................................... 884
  Bulk Loading for Sybase IQ ................................................................... 884
  Locking in the Sybase IQ Interface ......................................................... 885
  Naming Conventions for Sybase IQ ........................................................ 887
  Data Types for Sybase IQ ..................................................................... 887
Chapter 33 • SAS/ACCESS Interface to Teradata ................................. 891
Introduction to SAS/ACCESS Interface to Teradata .......................... 892
LIBNAME Statement Specifics for Teradata .................................... 893
Data Set Options for Teradata ....................................................... 894
SQL Pass-Through Facility Specifics for Teradata .............................. 901
Temporary Table Support for Teradata .............................................. 904
Passing SAS Functions to Teradata ................................................ 904
Passing Joins to Teradata ............................................................. 905
Maximizing Teradata Load and Read Performance .............................. 905
Autopartitioning Scheme for Teradata (Legacy) ................................. 914
Teradata Processing Tips for SAS Users .......................................... 918
Using the Teradata Wallet Feature ............................................... 922
Locking in the Teradata Interface .................................................. 924
Naming Conventions for Teradata .................................................. 929
Data Types for Teradata ............................................................... 931
Temporal Data Types for Teradata .................................................. 934

Chapter 34 • SAS/ACCESS Interface to Vertica ................................. 939
Introduction to SAS/ACCESS Interface to Vertica .............................. 939
LIBNAME Statement Specifics for Vertica ........................................ 940
Data Set Options for Vertica ......................................................... 943
SQL Pass-Through Facility Specifics for Vertica ................................. 945
Understanding Vertica Update and Delete Rules ................................. 947
Passing SAS Functions to Vertica .................................................... 947
Passing Joins to Vertica ............................................................... 948
Locking in the Vertica Interface ....................................................... 949
Naming Conventions for Vertica ...................................................... 950
Data Types for Vertica ................................................................. 951

PART 4  Appendixes 953

Appendix 1 • ACCESS Procedure .................................................. 955
Overview: ACCESS Procedure ....................................................... 955
DBMS Specifics: ACCESS Procedure .............................................. 957
Syntax: ACCESS Procedure .......................................................... 957
Examples: ACCESS Procedure ..................................................... 960

Appendix 2 • CV2VIEW Procedure ................................................. 975
Overview: CV2VIEW Procedure ..................................................... 975
Syntax: CV2VIEW Procedure ......................................................... 976
Examples: CV2VIEW Procedure ..................................................... 980

Appendix 3 • DBLOAD Procedure .................................................. 985
Overview: DBLOAD Procedure ..................................................... 985
DBMS Specifics: DBLOAD Procedure .............................................. 987
Syntax: DBLOAD Procedure .......................................................... 987
Example: Append a Data Set to a DBMS Table ................................. 997

Recommended Reading ............................................................... 999
Glossary .............................................................................. 1001
Index .................................................................................. 1009
What's New in SAS/ACCESS 9.4 for Relational Databases

Overview

Here are the new features and enhancements in 9.4 for SAS/ACCESS for Relational Databases.

- PostgreSQL, SAP HANA, and Vertica are new SAS/ACCESS interfaces.
- The HP Neoview interface is no longer supported.
- SAS/ACCESS no longer supports HP-UX or Linux x86 operating environments. For changes to other operating environments for specific SAS/ACCESS interfaces, see SAS/ACCESS Features by Host for your interface.
- documentation enhancements

In the first maintenance release for SAS 9.4, the BL_ESCAPE= data set option is new.

In the second maintenance release for SAS 9.4, Cloudera Impala is a new SAS/ACCESS interface.

In the third maintenance release for SAS 9.4, these features are new or enhanced.

- HAWQ is a new SAS/ACCESS interface.
- The format for specifying octal codes using the BL_ESCAPE= data set option has changed. Specifying them with a preceding ‘E’ is no longer supported. For example, specify E'\24' instead as the three-digit octal code, \024.
- A new section about National Language Support (NLS) considerations, issues, and limitations was added.

System Option: SQLGENERATION=

In the third maintenance release for SAS 9.4, the default value of the SQLGENERATION= system option is now (NONE DBMS='TERADATA DB2 ORACLE NETEZZA ASTER GREENPLM HADOOP SAPHANA IMPALA HAWQ').
SAS/ACCESS Interface to Aster

Using the new BL_MAPFILE= data set option, you can specify a map file to load or export tables.

In the first maintenance release for SAS 9.4, the Aster interface now supports AIX and Solaris for SPARC.

In the third maintenance release for SAS 9.4, the following options were added:

- POST_STMT_OPTS= data set and LIBNAME options
- POST_TABLE_OPTS= data set option
- PRE_STMT_OPTS= data set option
- PRE_TABLE_OPTS= data set option

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts

With the new PROGRAM_NAME= LIBNAME option, you can specify the string to use as the application identifier for DB2 monitoring.

In the first maintenance release for SAS 9.4, these options are new.

- You can use the new PRESERVE_USER= LIBNAME option to retain the case sensitivity of the value for the USER= option.
- The CHAR_AS_BINARY= data set option can preserve all character columns as binary values when reading data from the database.

In the second maintenance release for SAS 9.4, the maximum length for column names is now 32 characters (rather than 30).

In the third maintenance release for SAS 9.4, the following options were added:

- POST_STMT_OPTS= data set and LIBNAME options
- POST_TABLE_OPTS= data set option
- PRE_STMT_OPTS= data set option
- PRE_TABLE_OPTS= data set option

SAS/ACCESS Interface to DB2 under z/OS

In the first maintenance release for SAS 9.4, the ALLOWED_SQLCODES= LIBNAME option lets you specify SQL warnings to suppress during preparation and execution.

In the second maintenance release for SAS 9.4, the following features are new or updated:
- SAS/ACCESS Interface to DB2 under z/OS supports the BLOB, CLOB, and DBCLOB data types.
- The DB2DEBUG system option is no longer supported. To view the same database actions in the log, specify the SASTRACE= system option instead as SASTRACE=',,d'.

In the third maintenance release for SAS 9.4, the following options were added:
- POST_STMT_OPTS= data set and LIBNAME options
- POST_TABLE_OPTS= data set option
- PRE_STMT_OPTS= data set option
- PRE_TABLE_OPTS= data set option

---

**SAS/ACCESS Interface to Greenplum**

The new **BL_DATAFILE_EXISTS=** data set option lets you load a Greenplum table from an existing data set.

In the second maintenance release for SAS 9.4, SAS/ACCESS Interface to Greenplum supports the following options:
- DBSLICE= data set option
- DBSLICEPARAM= LIBNAME option, data set option, and system option

In the third maintenance release for SAS 9.4, these features are new or enhanced:
- You can pass the ** operator, COT function, and COMPRESS function to Greenplum via the SQL pass-through facility.
- The **POST_STMT_OPTS=** data set and LIBNAME options are available.
- The **POST_TABLE_OPTS=** data set option is available.
- The **PRE_STMT_OPTS=** data set option is available.
- The **PRE_TABLE_OPTS=** data set option is available.

---

**SAS/ACCESS Interface to Hadoop**

In SAS 9.4, this interface has these enhancements.
- HDFS_TEMPDIR=, HDFS_METADIR=, HDFS_DATADIR=, SUBPROTOCOL=, HIVE_PRINCIPAL=, and HDFS_PRINCIPAL= are new LIBNAME connection options.
- The **HDFS_METADIR=** LIBNAME connection option signals SAS/ACCESS to access HDFS data using metadata that is defined in SAS instead of using Hive or HiveServer2.
- Default values for the **SQLGENERATION=** LIBNAME and system options have changed.
You can use the COLUMN_DELIMITER= data set option to specify a single character as a column delimiter.

In the second maintenance release for SAS 9.4, these features are new or enhanced.

- You can specify JDBC connection properties that override the default connection properties using the PROPERTIES= LIBNAME option.
- The default value for the SUBPROTOCOL= connection option is HiveServer2.
- The CFG= option for the LIBNAME= statement is deprecated. Instead of specifying the CFG= option, you define the SAS environment variable SAS_HADOOP_CONFIG_PATH to set the location of the Hadoop cluster configuration files.
- The SAS environment variable SAS_HADOOP_CONFIG_PATH can be defined and set to the location of the Hadoop configuration files. Hadoop configuration files include core-site.xml, hdfs-site.xml, hive-site.xml, mapred-site.xml, and, if applicable, yarn-site.xml.

**Note:** If you are using MapR, no hdfs-site.xml file is required in the directory.

- You can create and append to Hive tables through the WebHDFS service instead of the HDFS service by setting the SAS environment variable SAS_HADOOP_RESTFUL.
- You can create and append to non-textual Hive table formats, such as SEQUENCEFILE, RCFILE, ORC, and PARQUET.
- SAS supports the Hive data types TIMESTAMP, DATE, and VARCHAR for Hive 0.12, and CHAR for Hive 0.13.
- HiveServer2 is supported. Hive continues to be supported in this release.
- SAS supports Hive authorization and authentication using IBM InfoSphere BigInsights 2.1.

In the third maintenance release for SAS 9.4, these features are new or enhanced.

- You can use the new CONFIG= and CONFIGDIR= LIBNAME and data set options to define the name and location of bulk-load configuration files.
- Support for DBCREATE_TABLE_OPTS= LIBNAME option is new.
- The following data set options are available to specify options in the CREATE TABLE statement for the SQL procedure: POST_STMT_OPTS=, POST_TABLE_OPTS=, PRE_STMT_OPTS=, and PRE_TABLE_OPTS=. The POST_STMT_OPTS= LIBNAME option is also available.
- The TRANSCODE_FAIL= LIBNAME and data set options are new.
- Support for SQLGENERATION= LIBNAME and system options is new.
- The NUMTASKS= LIBNAME option is no longer supported.
- BINARY and DECIMAL data type support is new.
- The new SQOOP procedure lets you access Apache Sqoop within a SAS session so that you can transfer data between a database and HDFS. For more information about the SQOOP procedure, see *Base SAS Procedures Guide*. 


SAS/ACCESS Interface to HAWQ

In the third maintenance release for SAS 9.4, SAS/ACCESS Interface to HAWQ is a new engine that is supported for SAS/ACCESS. This interface provides direct, transparent access to HAWQ through LIBNAME statements and the SQL pass-through facility. You can use various LIBNAME statement options and data set options that the LIBNAME engine supports to control the data that is returned to SAS.

SAS/ACCESS Interface to Impala

In the second maintenance release for SAS 9.4, SAS/ACCESS Interface to Cloudera Impala is a new database engine that runs on specific UNIX platforms. This interface provides direct, transparent access to Impala through LIBNAME statements and the SQL pass-through facility. You can use various LIBNAME statement options and data set options that the LIBNAME engine supports to control the data that is returned to SAS.

In the second maintenance release for SAS 9.4, you can perform these tasks:

- Configure a required set of Hadoop JAR files. Indicate the JAR file location with the SAS_HADOOP_JAR_PATH environment variable.
- Specify the Kerberos principal for the Impala server with the IMPALA_PRINCIPAL= or HDFS_PRINCIPAL= LIBNAME option or with the HDFS_PRINCIPAL= data set option.
- Specify column names to use as partition keys to create fact tables with the PARTITIONED_BY= data set option.

In the second maintenance release for SAS 9.4, you can use an environment variable and these options for bulk loading:

- Set the SAS_HADOOP_RESTFUL environment variable to 1 when you start SAS to use WebHDFS for bulk loading.
- Use BL_HOST=, BL_PORT=, and BULKLOAD= LIBNAME options for bulk loading. You must set BULKLOAD=YES to perform bulk loading of data within a libref. All other bulk-loading LIBNAME options are optional.
- Use BL_DATAFILE=, BL_DELETE_DATAFILE=, BL_HOST=, BL_PORT=, and BULKLOAD= data set options for bulk loading.
  You must set BULKLOAD=YES to enable bulk loading for a data set if it is not already enabled for the libref. All other data set options for bulk loading are optional.

In the third maintenance release for SAS 9.4, these features are new or enhanced:

- Use IMPALA in the LIBNAME statement to specify this engine.
- You can use the new CONFIG= and CONFIGDIR= LIBNAME and data set options to define the name and location of bulk-load configuration files.
- Support for DBCLIENT_MAXgetBytes= and DBSERVER_MAXBYTES= LIBNAME options is new.
The new DRIVER_VENDOR= LIBNAME option lets you indicate the name of the vendor of your specific Hadoop distribution.

The following data set options are available to specify options in the CREATE TABLE statement for the SQL procedure: POST_STMT_OPTS=, POST_TABLE_OPTS=, PRE_STMT_OPTS=, and PRE_TABLE_OPTS=. The POST_STMT_OPTS= LIBNAME option is also available.

Support for SQLGENERATION= LIBNAME and system options is new.

SAS/ACCESS Interface to Informix

In the third maintenance release for SAS 9.4, the following options were added:

- POST_STMT_OPTS= data set and LIBNAME options
- POST_TABLE_OPTS= data set option
- PRE_STMT_OPTS= data set option
- PRE_TABLE_OPTS= data set option

SAS/ACCESS Interface to Microsoft SQL Server

In the third maintenance release for SAS 9.4, these features are new or enhanced:

- Support was added for Microsoft Windows for x86 and for Microsoft Windows for x64 platforms. For more information, see “SAS/ACCESS Interface to Microsoft SQL Server: Supported Features” on page 83.

- The following data set options are available to specify options in the CREATE TABLE statement for the SQL procedure: POST_STMT_OPTS=, POST_TABLE_OPTS=, PRE_STMT_OPTS=, and PRE_TABLE_OPTS=. The POST_STMT_OPTS= LIBNAME option is also available.

SAS/ACCESS Interface to MySQL

In the third maintenance release for SAS 9.4, these features are new or enhanced:

- You can pass the ATAN2 and TRANWRD functions to MySQL via the SQL pass-through facility.

- The list of functions that can be passed automatically to MySQL and the list of functions that require SQL_FUNCTIONS=ALL have been updated. For more information, see “Passing SAS Functions to MySQL” on page 704.
SAS/ACCESS Interface to Netezza

These features are new or enhanced.

- The new SYNONYMS= LIBNAME option shows synonyms, tables, views, or materialized views.
- In the second maintenance release for SAS 9.4, support for the Netezza ST_GEOMETRY and VARBINARY data types was added, also support for multiple Netezza schemas.

In the third maintenance release for SAS 9.4, the following options were added:

- POST_STMT_OPTS= data set and LIBNAME options
- POST_TABLE_OPTS= data set option
- PRE_STMT_OPTS= data set option
- PRE_TABLE_OPTS= data set option

SAS/ACCESS Interface to ODBC

Use the new DATETIME2= LIBNAME or data set option to specify the scale for the timestamp literal for Microsoft SQL Server 2008 and the native Microsoft driver.

SAS/ACCESS Interface to OLE DB

In SAS 9.4, The BOOL_VAL environment variable is new.

In the third maintenance release for SAS 9.4, these features have been added or enhanced:

- The PRESERVE_GUID= LIBNAME option was added.
- The COMPRESS function can be passed to OLE DB only when SQL_FUNCTIONS= is set to ALL.

SAS/ACCESS Interface to Oracle

In the first maintenance release for SAS 9.4, the new default value for the OR_BINARY_DOUBLE= LIBNAME option is YES.

The second maintenance release for SAS 9.4 has these enhancements:

- You can use the BL_API_BULKLOAD= data set option to perform bulk loading using the Oracle Direct Path API instead of the Oracle SQL*Loader utility.
Starting with Oracle 12c, the default data type for character variables in SAS output has been changed to either VARCHAR2 or CLOB, depending on the length of the variable. When the NOTRANSCODE attribute is set, the default data type can be either RAW or BLOB, depending on the length of the variable.

In the third maintenance release for SAS 9.4, the following options were added:

- POST_STMT_OPTS= data set and LIBNAME options
- POST_TABLE_OPTS= data set option
- PRE_STMT_OPTS= data set option
- PRE_TABLE_OPTS= data set option

SAS/ACCESS Interface to PostgreSQL

SAS/ACCESS Interface to PostgreSQL is a new database engine. It provides direct, transparent access to PostgreSQL through LIBNAME statements and the SQL pass-through facility. You can use various LIBNAME statement options and data set options that the LIBNAME engine supports to control the data that is returned to SAS.

In the first maintenance release for SAS 9.4, these features are new or enhanced:

- These data set options are new: BL_ESCAPE=, BL_FORMAT=, BL_NULL=, and BL_QUOTE=.
- The default value for the PORT= connection option for the LIBNAME statement is 5432.

In the third maintenance release for SAS 9.4, these features were added or enhanced:

- The COMPRESS and COT functions can be passed automatically to PostgreSQL via the pass-through facility.
- The POST_STMT_OPTS= data set and LIBNAME options are available.
- The POST_TABLE_OPTS= data set option is available.
- The PRE_STMT_OPTS= data set option is available.
- The PRE_TABLE_OPTS= data set option is available.

SAS/ACCESS Interface to SAP HANA

In the July 2013 release of SAS 9.4, SAS/ACCESS Interface to SAP HANA is a new database engine. It provides direct, transparent access to SAP HANA through LIBNAME statements and the SQL pass-through facility. You can use various LIBNAME statement options and data set options that the LIBNAME engine supports to control the data that is returned to SAS.

In the second maintenance release for SAS 9.4, the saphana LIBNAME engine name was added.

In the third maintenance release for SAS 9.4, these features are new or enhanced.
• SAS/ACCESS generates a default statement to read SAP HANA analytic views based on the metadata.

• Support for the SQLGENERATION= LIBNAME and system options is new.

• You can automatically pass down the SAS ** (POWER(base, exponent)) and COT functions to SAP HANA.

• PARMSTRING= and PARMDEFAULT= LIBNAME options and PARMSTRING= and PARMDEFAULT= data set options are available.

SAS/ACCESS Interface to Sybase

In the third maintenance release for SAS 9.4, the following options were added:

• POST_STMT_OPTS= data set and LIBNAME options

• POST_TABLE_OPTS= data set option

• PRE_STMT_OPTS= data set option

• PRE_TABLE_OPTS= data set option

SAS/ACCESS Interface to Sybase IQ

In the third maintenance release for SAS 9.4, the following options were added:

• POST_STMT_OPTS= data set and LIBNAME options

• POST_TABLE_OPTS= data set option

• PRE_STMT_OPTS= data set option

• PRE_TABLE_OPTS= data set option

SAS/ACCESS Interface to Teradata

These features are new or enhanced.

• You can set database client encoding to match SAS encoding with the DBCLIENT_MAX_BYTES= LIBNAME option.

• Teradata data types and character-set metadata are transferred from input to output for DATA step processing. Set the new SAS_DBMS_AUTOMETADATA= LIBNAME option to YES, or set the new SAS_DBMS_AUTOMETADATA= environment variable for an entire SAS session.

• The TPT_MAX_SESSIONS= data set option has a new default value. When you set this option, it has precedence over any value set for the default, for the new TPT_MAX_SESSIONS= LIBNAME option, and for the new SAS_TPT_MAX_SESSIONS environment variable.

• TPT messages are now displayed in PROC SQL.
• The DATEPART function is now passed down by default.

In the third maintenance release, these features are new or enhanced:

• SAS/ACCESS supports object names that contain up to 32 characters for users who use Teradata 14.10 or higher.
• Support was added for the Teradata Wallet security feature.
• The POST_STMT_OPTS= data set and LIBNAME options are available.
• The POST_TABLE_OPTS= data set option is available.
• The PRE_STMT_OPTS= data set option is available.
• The PRE_TABLE_OPTS= data set option is available.
• The TPT_MIN_SESSIONS=LIBNAME option is available.
• The SUM4 function was removed from the list of functions that can be passed to Teradata. The TRIM function can be passed automatically and does not require SQL_FUNCTIONS=ALL. The LENGTH function does require SQL_FUNCTIONS=ALL.

SAS/ACCESS Interface to Vertica

SAS/ACCESS Interface to Vertica is a new database engine for SAS 9.4. It provides direct, transparent access to Vertica through LIBNAME statements and the SQL pass-through facility. You can use various LIBNAME statement options and data set options that the LIBNAME engine supports to control the data that is returned to SAS.

In the third maintenance release for SAS 9.4, the following options were added:

• POST_STMT_OPTS= data set and LIBNAME options
• POST_TABLE_OPTS= data set option
• PRE_STMT_OPTS= data set option
• PRE_TABLE_OPTS= data set option

Documentation Enhancements

This document includes these new and enhanced items.

• In the third maintenance release for SAS 9.4, the following additions or updates have been made throughout the documentation:
  • Information about the HDMD Procedure was moved from this document to the Base SAS Procedures Guide.
  • Information about supported data types has been consolidated into a single section for each DBMS.
  • The SQL pass-through function MOD might yield different results from SAS if non-integer arguments are specified. See the documentation for your DBMS for details.
Aster:
- You can use the `BL_DATAFILE_PATH=` data set option to specify a path for creating a flat file for bulk loading.
- Beginning in the second maintenance release for SAS 9.4, you can use the `BULKUNLOAD=` LIBNAME and data set options to rapidly retrieve records from a data set.

DB2 under z/OS: In the third maintenance release for SAS 9.4, the following documentation additions or updates have been made:
- The `DEGREE=`, `LOCATION=`, and `READBUFF=` connection arguments were added to the list of those that are supported in the `CONNECTION` statement for the SQL pass-through facility.
- The list of default SAS formats for various data types has been updated.
- When you store temporal data and the corresponding history data, you must store the history data in a separate tablespace.

MySQL: Bulk loading is supported.

Netezza: To set `BULKUNLOAD=YES`, you must have Create External Table permission.

Oracle:
- Beginning in the first maintenance release for SAS 9.4, you can indicate fixed-width encoding with the new `DBCLIENT_ENCODING_FIXED=` and `DBSERVER_ENCODING_FIXED=` LIBNAME options.
- Beginning in SAS 9.4, you can generate identity columns during table creation by using the `OR.IDENTITY_COLS=` data set option.

PostgreSQL: In the third maintenance release for SAS 9.4, a new section, “Working with Long Character Values in PostgreSQL,” was added.

SAP HANA:
- In the second maintenance release for SAS 9.4, the documentation explains how to work with SAP HANA geospatial data.
- In the third maintenance release for SAS 9.4, a new section, “SAP HANA Schema Flexibility,” was added.
Accessibility

For information about the accessibility of this product, see Accessibility Features of the Windowing Environment for SAS 9.4 at support.sas.com.
Part 1

Concepts

Chapter 1
Overview of SAS/ACCESS Interface to Relational Databases ............ 3

Chapter 2
SAS Names and Support for DBMS Names ............................. 13

Chapter 3
Data Integrity and Security ............................................. 27

Chapter 4
Performance Considerations ............................................ 37

Chapter 5
Optimizing Your SQL Usage ........................................... 43

Chapter 6
Threaded Reads ............................................................ 53

Chapter 7
National Language Support .............................................. 63

Chapter 8
How SAS/ACCESS Works ............................................... 65

Chapter 9
In-Database Processing with SAS/ACCESS ........................... 71
Chapter 1
Overview of SAS/ACCESS Interface to Relational Databases

About This Document

This document provides conceptual, reference, and usage information for SAS/ACCESS software for data sources for relational database management systems (DBMSs), data warehouse appliances, and distributed systems. The information in this document applies to all data sources that SAS/ACCESS software supports.

The availability and behavior of SAS/ACCESS features vary from one interface to another. Therefore, consult the information in both the general and DBMS-specific sections of this document when working with your particular SAS/ACCESS interface. Also, refer to the SAS system requirements and configuration guide documents that are available at http://support.sas.com.

This document is intended for applications programmers and end users with these skills.

• They are familiar with the basics of their DBMS or data warehouse appliance and its Structured Query Language (SQL).

• If they are using Hadoop, they are familiar with Hadoop and the Hadoop Distributed File System (HDFS). They have knowledge of fundamental Hadoop library and processing using HDFS. They are also familiar with the basics of Hadoop, Hive, and HiveQL. For details, see the Apache Hadoop and Hive websites at http://hadoop.apache.org and https://cwiki.apache.org/confluence/display/Hive.
• They know how to use their operating environment.
• They can use basic SAS commands and statements.

Database administrators might also want to read this document to understand how to implement and administer specific interfaces.

## Methods for Accessing Relational Database Data

SAS/ACCESS Interface to Relational Databases is a family of interfaces—each licensed separately—with which you can interact with data in other vendor databases from within SAS. SAS/ACCESS provides these methods for accessing relational DBMS data.

- To assign SAS librefs to DBMS objects such as schemas and databases, you can use the **LIBNAME statement**. After you associate a database with a libref, you can use a SAS two-level name to specify any table or view in the database. You can then work with the table or view as you would with a SAS data set.

- To interact with a data source using its native SQL syntax without leaving your SAS session, you can use the **SQL pass-through facility**. SQL statements are passed directly to the data source for processing.

- For indirect access to DBMS data, you can use **ACCESS** and **DBLOAD** procedures. Although SAS still supports these legacy procedures for specific DBMSs and environments, they are no longer the recommended method for accessing DBMS data.

For information about when to use each method, see “Selecting a SAS/ACCESS Method” on page 4.

Not all SAS/ACCESS interfaces support all of these features. To determine which features are available in your environment, see SAS/ACCESS Features by Host on page 77.

## Selecting a SAS/ACCESS Method

### Methods for Accessing DBMS Tables and Views

You can often complete a task in SAS/ACCESS in several ways. For example, you can access DBMS tables and views by using the **LIBNAME statement** or the **SQL pass-through facility**. Before processing complex or data-intensive operations, you might want to test several methods first to determine the most efficient one for your particular task.

### SAS/ACCESS LIBNAME Statement Advantages

You should use the SAS/ACCESS LIBNAME statement for the fastest and most direct method of accessing your DBMS data except when you need to use SQL that is not ANSI-standard. ANSI-standard SQL is required when you use the SAS/ACCESS library engine in the SQL procedure. However, the SQL pass-through facility accepts all SQL extensions that your DBMS provides.

Here are the advantages of using the SAS/ACCESS LIBNAME statement.
• Significantly fewer lines of SAS code are required to perform operations on your DBMS. For example, a single LIBNAME statement establishes a connection to your DBMS, lets you specify how data is processed, and lets you easily view your DBMS tables in SAS.

• You do not need to know the SQL language of your DBMS to access and manipulate data on your DBMS. You can use such SAS procedures as PROC SQL or DATA step programming on any libref that references DBMS data. You can read, insert, update, delete, and append data. You can also create and drop DBMS tables by using SAS syntax.

• The LIBNAME statement gives you more control over DBMS operations such as locking, spooling, and data type conversion through the use of LIBNAME and data set options.

• The engine can optimize processing of joins and WHERE clauses by passing them directly to the DBMS, which takes advantage of the indexing and other processing capabilities of your DBMS. For more information, see Optimizing Your SQL Usage on page 43.

• The engine can pass some functions directly to the DBMS for processing.

**SQL Pass-Through Facility Advantages**

Here are the advantages of using the SQL pass-through facility.

• You can use SQL pass-through facility statements so that the DBMS can optimize queries, particularly when you join tables. The DBMS optimizer can take advantage of indexes on DBMS columns to process a query more quickly and efficiently.

• SQL pass-through facility statements let the DBMS optimize queries when queries have summary functions. Summary functions include AVG, COUNT, GROUP BY clauses, or columns that are created by expressions, such as those that use the COMPUTED function. The DBMS optimizer can use indexes on DBMS columns to process queries more rapidly.

• On some DBMSs, you can use SQL pass-through facility statements with SAS/AF applications to handle transaction processing of DBMS data. Using a SAS/AF application gives you complete control of COMMIT and ROLLBACK transactions. SQL pass-through facility statements give you better access to DBMS return codes.

• The SQL pass-through facility accepts all extensions to ANSI SQL that your DBMS provides.

**SAS/ACCESS Features for Common Tasks**

Here is a list of tasks and the features that you can use to accomplish them.

<table>
<thead>
<tr>
<th>Task</th>
<th>SAS/ACCESS Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read DBMS tables or views</td>
<td>LIBNAME statement</td>
</tr>
<tr>
<td></td>
<td>SQL pass-through facility</td>
</tr>
<tr>
<td></td>
<td>View descriptors</td>
</tr>
<tr>
<td>Task</td>
<td>SAS/ACCESS Features</td>
</tr>
<tr>
<td>------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Create DBMS objects, such as tables</td>
<td>LIBNAME statement*</td>
</tr>
<tr>
<td></td>
<td>DBLOAD procedure</td>
</tr>
<tr>
<td></td>
<td>SQL pass-through facility EXECUTE statement</td>
</tr>
<tr>
<td>Update, delete, or insert rows into DBMS tables</td>
<td>LIBNAME statement*</td>
</tr>
<tr>
<td></td>
<td>View descriptors**</td>
</tr>
<tr>
<td></td>
<td>SQL pass-through facility EXECUTE statement</td>
</tr>
<tr>
<td>Append data to DBMS tables</td>
<td>DBLOAD procedure with APPEND option</td>
</tr>
<tr>
<td></td>
<td>LIBNAME statement and APPEND procedure*</td>
</tr>
<tr>
<td></td>
<td>SQL pass-through facility EXECUTE statement</td>
</tr>
<tr>
<td></td>
<td>SQL pass-through facility INSERT statement</td>
</tr>
<tr>
<td>List DBMS tables</td>
<td>LIBNAME statement and SAS Explorer window*</td>
</tr>
<tr>
<td></td>
<td>LIBNAME statement and DATASETS procedure*</td>
</tr>
<tr>
<td></td>
<td>LIBNAME statement and CONTENTS procedure*</td>
</tr>
<tr>
<td></td>
<td>LIBNAME statement and SQL procedure dictionary tables*</td>
</tr>
<tr>
<td>Delete DBMS tables or views</td>
<td>LIBNAME statement and SQL procedure DROP TABLE statement*</td>
</tr>
<tr>
<td></td>
<td>LIBNAME statement and DATASETS procedure DELETE statement*</td>
</tr>
<tr>
<td></td>
<td>DBLOAD procedure with SQL DROP TABLE statement</td>
</tr>
<tr>
<td></td>
<td>SQL pass-through facility EXECUTE statement</td>
</tr>
</tbody>
</table>

* LIBNAME statement refers to the SAS/ACCESS LIBNAME statement.
** View descriptors refer to view descriptors that are created in the ACCESS procedure.

---

**SAS Views of DBMS Data**

SAS/ACCESS lets you create a SAS view of data that exists in a DBMS. A *SAS data view* defines a virtual data set that is named and stored for later use. A view contains no data, but rather describes data that is stored elsewhere. Here are the types of SAS data views.

DATA step views
- stored, compiled DATA step programs.
SQL views are stored query expressions that read data values from their underlying files, which can include SAS data files, SAS/ACCESS views, DATA step views, other SQL views, or relational database data.

SAS/ACCESS views (also called view descriptors) describe data that is stored in DBMS tables. This is no longer a recommended method for accessing relational DBMS data. To convert existing view descriptors into SQL views, use the CV2View procedure on page 975.

You can use all types of views as inputs into DATA steps and procedures. You can specify views in queries as if they were tables. A view derives its data from the tables or views that are listed in its FROM clause. The data accessed by a view is a subset or superset of the data in its underlying table(s) or view(s).

You can use SQL views and SAS/ACCESS views to update their underlying data if one of the following is true:

- The view is based on only one DBMS table.
- The view is based on a DBMS view that is based on only one DBMS table, and the view has no calculated fields.

You cannot use DATA step views to update the underlying data; you can use them only to read the data.

Your options for creating a SAS view of DBMS data are determined by the SAS/ACCESS feature that you are using to access the DBMS data. This table lists the recommended methods for creating SAS views.

<table>
<thead>
<tr>
<th>Feature for Accessing DBMS Data</th>
<th>SAS View Technology to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS/ACCESS LIBNAME statement</td>
<td>SQL view or DATA step view of the DBMS table</td>
</tr>
<tr>
<td>SQL pass-through facility</td>
<td>SQL view with CONNECTION TO component</td>
</tr>
</tbody>
</table>

Choosing Your Degree of Numeric Precision

Factors That Can Cause Calculation Differences

Different factors affect numeric precision. This issue is common for many people, including SAS users. Though computers and software can help, you are limited in how precisely you can calculate, compare, and represent data. Therefore, only those people who generate and use data can determine the exact degree of precision that meets their enterprise needs.

As you decide the degree of precision that you want, you need to consider that these system factors can cause calculation differences:

- hardware limitations
- differences among operating systems
- different software or different versions of the same software
• different DBMSs

These factors can also cause differences:
• the use of finite number sets to represent infinite real numbers
• how numbers are stored, because storage sizes can vary

You also need to consider how conversions are performed on, between, or across any of these system or calculation factors.

Examples of Problems That Result in Numeric Imprecision

Overview
Depending on the degree of precision that you want, calculating the value of $r$ can result in a tiny residual in a floating-point unit. When you compare the value of $r$ to 0.0, you might find that $r \neq 0.0$. The numbers are very close but not equal. This type of discrepancy in results can stem from problems in representing, rounding, displaying, and selectively extracting data.

Representing Data
Some numbers can be represented exactly, but others cannot. As shown in this example, the number 10.25, which terminates in binary, can be represented exactly.

```sas
data x;
  x=10.25;
  put x hex16.;
run;
```

The output from this DATA step is an exact number: 4024800000000000. However, the number 10.1 cannot be represented exactly, as this example shows.

```sas
data x;
  x=10.1;
  put x hex16.;
run;
```

The output from this DATA step is an inexact number: 4024333333333333.

Rounding Data
As this example shows, rounding errors can result from platform-specific differences. No solution exists for such situations.

```sas
data x;
  x=10.1;
  put x hex16.;
  y=100000;
  newx=(x+y)-y;
  put newx hex16.;
run;
```

In Windows and Linux environments, the output from this DATA step is 4024333333333333 (8/10-byte hardware double). In the Solaris x64 environment, the output is 4024333333334000 (8/8-byte hardware double).

Displaying Data
For certain numbers such as $x.5$, the precision of displayed data depends on whether you round up or down. Low-precision formatting (rounding down) can produce different
results on different platforms. In this example, the same high-precision (rounding up)
result occurs for X=8.3, X=8.5, or X=hex16. However, a different result occurs for
X=8.1 because this number does not yield the same level of precision.

data;
  x=input('C047DFFFFFFFFFFFFF', hex16.);
  put x= 8.1 x= 8.3 x= 8.5 x= hex16.;
run;

Here is the output under Windows or Linux (high-precision formatting).
  x=-47.8
  x=-47.750 x=-47.7500
  x=C047DFFFFFFFFFFFFF

Here is the output under Solaris x64 (low-precision formatting).
  x=-47.7
  x=-47.750 x=-47.7500
  x=C047DFFFFFFFFFFFFF

To fix the problem that this example illustrates, you must select a number that yields the
next precision level—in this case, 8.2.

**Selectively Extracting Data**

Results can also vary when you access data that is stored on one system by using a client
on a different system. This example illustrates running a DATA step from a Windows
client to access SAS data in the z/OS environment.

data z(keep=x);
  x=5.2;
  output;
  y=1000;
  x=(x+y)-y; /*almost 5.2 */
  output;
run;

proc print data=z;
run;

Here is the output that this DATA step produces.

<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.2</td>
</tr>
<tr>
<td>2</td>
<td>5.2</td>
</tr>
</tbody>
</table>

The next example illustrates the output that you receive when you execute the DATA
step interactively under Windows or z/OS.

data z1;
  set z(where=(x=5.2));
run;

Here is the corresponding z/OS output.

NOTE: There were 1 observations read from the data set WORK.Z.
WHERE x=5.2;
NOTE: The data set WORK.Z1 has 1 observations and 1 variables.
The DATA statement used 0.00 CPU seconds and 14476K.

In the above example, the expected count was not returned correctly under z/OS because
the imperfection of the data and finite precision are not taken into account. You cannot
use equality to obtain a correct count because it does not include the “almost 5.2” cases in that count. To obtain the correct results under z/OS, you must run this DATA step:

data z1;
  set z(where=(compfuzz(x,5.2,1e-10)=0));
run;

Here is the z/OS output from this DATA step.

NOTE: There were 2 observations read from the data set WORK.Z.
WHERE COMPFUZZ(x, 5.2, 1E-10)=0;
NOTE: The data set WORK.Z1 has 2 observations and 1 variables.

Your Options When Choosing the Degree of Precision That You Need

After you determine the degree of precision that your enterprise needs, you can refine your software. You can use macros, sensitivity analyses, or fuzzy comparisons such as extractions or filters to extract data from databases or from different versions of SAS.

If you are running SAS 9.2, use the COMPFUZZ (fuzzy comparison) function. Otherwise, use this macro.

/*****************************************************************************/
/* This macro defines an EQFUZZ operator. The subsequent DATA step shows */
/* how to use this operator to test for equality within a certain tolerance. */
/*****************************************************************************/
%macro eqfuzz(var1, var2, fuzz=1e-12);
  abs((&var1 - &var2) / &var1) < &fuzz
%mend;

data _null_;  
  x=0;  
  y=1;  
  do i=1 to 10;  
    x+0.1;  
    if x=y then put 'x exactly equal to y';  
    else if %eqfuzz(x,y) then put 'x close to y';  
    else put 'x nowhere close to y';  
  end;  
run;

When you read numbers in from an external DBMS that supports precision beyond 15 digits, you can lose that precision. You cannot do anything about this for existing databases. However, when you design new databases, you can set constraints to limit precision to about 15 digits. Alternatively, you can select a numeric DBMS data type to match the numeric SAS data type. For example, select the BINARY_DOUBLE type in Oracle (precise up to 15 digits) instead of the NUMBER type (precise up to 38 digits).

When you read numbers in from an external DBMS for noncomputational purposes, use the DBSASTYPE= data set option, as shown in this example.

libname ora oracle user=myusr1 password=mypwd1 path=mypath;
data sasdata;
  set ora.catalina2( dbastype= { c1='char(20)'} ) ;
run;

This option retrieves numbers as character strings and preserves precision beyond 15 digits. For details, see the DBSASTYPE= data set option on page 346.
References

See these resources for more detail about numeric precision, including variables that can affect precision.

- “Numerical Accuracy in SAS Software” in *SAS Language Reference: Concepts*

National Language Support

SAS/ACCESS provides National Language Support (NLS) in a variety of ways. These LIBNAME or SQL pass-through options allow for byte and character conversion and length calculation.

- ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=
- ADJUST_NCHAR_COLUMN_LENGTHS=
- DB_LENGTH_SEMANTICS_BYTE=
- DBCLIENT_MAX_BYTES=
- DBSERVER_MAX_BYTES=

These data types allow for more flexible adjustment of column lengths.

- BLOB
- CHAR
- CLOB
- DBCLOB
- NCHAR
- VARCHAR

For more NLS information, see these resources.

- For NLS limitations that are specific to Hive, see “Naming Conventions for SAS and Hive” on page 615.
- For additional NLS considerations, see the technical paper, “Processing Multilingual Data with the SAS 9.2 Unicode Server.” This paper is available at this URL: http://support.sas.com/resources/papers/92unicodesrvr.pdf
- For more comprehensive NLS information, see *SAS National Language Support (NLS): Reference Guide*.
## Chapter 2
SAS Names and Support for DBMS Names

<table>
<thead>
<tr>
<th>DBMS-Specific Naming Conventions</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS Naming Conventions</td>
<td>14</td>
</tr>
<tr>
<td>Length of Name</td>
<td>14</td>
</tr>
<tr>
<td>Case Sensitivity</td>
<td>14</td>
</tr>
<tr>
<td>SAS Name Literals</td>
<td>15</td>
</tr>
<tr>
<td>SAS/ACCESS Default Naming Behaviors</td>
<td>15</td>
</tr>
<tr>
<td>Modification and Truncation</td>
<td>15</td>
</tr>
<tr>
<td>ACCESS Procedure</td>
<td>15</td>
</tr>
<tr>
<td>DBLOAD Procedure</td>
<td>16</td>
</tr>
<tr>
<td>Renaming DBMS Data</td>
<td>16</td>
</tr>
<tr>
<td>Renaming SAS/ACCESS Tables</td>
<td>16</td>
</tr>
<tr>
<td>Renaming SAS/ACCESS Columns</td>
<td>16</td>
</tr>
<tr>
<td>Renaming SAS/ACCESS Variables</td>
<td>16</td>
</tr>
<tr>
<td>Options That Affect SAS/ACCESS Naming Behavior</td>
<td>17</td>
</tr>
<tr>
<td>Naming Behavior When Retrieving DBMS Data</td>
<td>17</td>
</tr>
<tr>
<td>Naming Behavior When Creating DBMS Objects</td>
<td>18</td>
</tr>
<tr>
<td>SAS/ACCESS Naming Examples</td>
<td>19</td>
</tr>
<tr>
<td>Replacing Unsupported Characters</td>
<td>19</td>
</tr>
<tr>
<td>Preserving Column Names</td>
<td>20</td>
</tr>
<tr>
<td>Preserving Table Names</td>
<td>20</td>
</tr>
<tr>
<td>Using DQUOTE=ANSI</td>
<td>22</td>
</tr>
<tr>
<td>Using Name Literals</td>
<td>24</td>
</tr>
<tr>
<td>Using DBMS Data to Create a DBMS Table</td>
<td>24</td>
</tr>
<tr>
<td>Using a SAS Data Set to Create a DBMS Table</td>
<td>25</td>
</tr>
</tbody>
</table>

## DBMS-Specific Naming Conventions

Some DBMSs allow case-sensitive names and names with special characters. As a result, keep these special considerations in mind when you use the names DBMS objects such as tables and columns with SAS/ACCESS features.

For information about how SAS handles your DBMS names, see the DBMS-specific reference section for your SAS/ACCESS interface.
SAS Naming Conventions

Length of Name

SAS naming conventions allow long names for SAS data sets and SAS variables. For example, MYDB.TEMP_EMPLOYEES_QTR4_2000 is a valid two-level SAS name for a data set.

Some SAS names can be up to 32 characters, depending on the SAS session encoding and the limits of the object name length of the DBMS. These SAS names can be up to 32 characters:

- members of SAS libraries, including SAS data sets, data views, catalogs, catalog entries, and indexes
- variables in a SAS data set
- macros and macro variables

These SAS language elements have a maximum length of eight characters:

- librefs and filerefs
- SAS engine names
- names of SAS/ACCESS access descriptors and view descriptors
- variable names in SAS/ACCESS access descriptors and view descriptors

For more information about SAS names, see SAS Language Reference: Concepts. For a complete description of SAS naming conventions, see *SAS Language Reference: Concepts*.

Case Sensitivity

When SAS encounters mixed-case or case-sensitive names in SAS code, SAS stores and displays the names as they are specified. If the SAS variables, Flight and dates, are defined in mixed case, here is an example.

```sas
input Flight $3. +3 dates date9.;
```

SAS then displays the variable names as defined, and the column headings appear as defined.

**Output 2.1 Mixed-Case Names Displayed in Output**

```
SAS System
Obs Flight dates
1 114 01MAR2000
2 202 01MAR2000
3 204 01MAR2000
```

Although SAS stores variable names as they are defined, it recognizes variables for processing without regard to case. For example, SAS processes these variables as
FLIGHT and DATES. Likewise, renaming the Flight variable to "flight" or "FLIGHT" would result in the same processing.

**SAS Name Literals**

A SAS name literal is a name token that is expressed as a quoted string that is followed by the letter n. By using name literals, you can use special characters or blanks that are not otherwise allowed in SAS names when you specify a SAS data set or variable. Name literals are especially useful for expressing database column and tables names that contain special characters.

Here are two examples of name literals.

```sas
data mydblib.'My Staff Table'n;

data Budget_for_1999;
  input '$ Amount Budgeted'n 'Amount Spent'n;
```

Name literals are subject to certain restrictions.

- You can use a name literal only for SAS variable and data set names, statement labels, and DBMS column and table names.
- You can use name literals only in a DATA step or in the SQL procedure.
- If a name literal contains any characters that are not allowed when VALIDVARNAMES=V7, you must set the system option to VALIDVARNAMES=ANY. For more information, see VALIDVARNAMES= on page 469.

---

**SAS/ACCESS Default Naming Behaviors**

**Modification and Truncation**

When SAS/ACCESS reads DBMS column names that contain characters that are not standard in SAS names, the default behavior is to replace an unsupported character with an underscore (_). (Nonstandard names include those with blank spaces or such special characters as @, #, % that are not allowed in SAS names.) For example, the DBMS column name Amount Budgeted$ becomes the SAS variable name Amount_Budgeted_.

When SAS/ACCESS encounters a DBMS name that exceeds 32 characters, it truncates the name.

After it has modified or truncated a DBMS column name, SAS appends a number to the variable name, if necessary, to preserve uniqueness. For example, DBMS column names MY$DEPT, My$Dept, and my$dept become SAS variable names MY_DEPT, MY_Dept0, and my_dept1.

**ACCESS Procedure**

If you attempt to use long names in the ACCESS procedure, you receive an error message advising you that long names are not supported. Long member names, such as access descriptor and view descriptor names, are truncated to eight characters. Long DBMS column names are truncated to 8-character SAS variable names within the SAS access descriptor. You can use the RENAME statement to specify 8-character SAS
variable names, or you can accept the default truncated SAS variable names that are assigned by the ACCESS procedure.

The ACCESS procedure converts DBMS object names to uppercase characters unless they are enclosed in quotation marks. Any DBMS objects that are given lowercase names when they are created, or whose names contain special or national characters, must be enclosed in quotation marks.

**DBLOAD Procedure**

You can use long member names, such as the name of a SAS data set that you want to load into a DBMS table, in the DBLOAD procedure DATA= option. However, if you attempt to use long SAS variable names, you receive an error message advising you that long variable names are not supported in the DBLOAD procedure. You can use the RENAME to rename the 8-character SAS variable names to long DBMS column names when you load the data into a DBMS table. You can also use the SAS data set option RENAME to rename the columns after they are loaded into the DBMS.

Most DBLOAD procedure statements convert lowercase characters in user-specified values and default values to uppercase. If your host or database is case sensitive and you want to specify a value that includes lowercase alphabetic characters (for example, a user ID or password), enclose the entire value in quotation marks. You must also put quotation marks around any value that contains special characters or national characters.

The only exception is the DBLOAD SQL statement. The DBLOAD SQL statement is passed to the DBMS exactly as you enter it with the case preserved.

**Renaming DBMS Data**

**Renaming SAS/ACCESS Tables**

You can rename DBMS tables and views using the CHANGE statement, as shown in this example.

```
proc datasets lib=x;
   change oldtable=newtable;
quit;
```

You can rename tables using this method for all SAS/ACCESS engines. However, if you change a table name, any view that depends on that table no longer works unless the view references the new table name.

**Renaming SAS/ACCESS Columns**

You can use the RENAME statement to rename the 8-character default SAS variable names to long DBMS column names when you load the data into a DBMS table. You can also use the SAS data set option RENAME= to rename the columns after they are loaded into the DBMS.

**Renaming SAS/ACCESS Variables**

You can use the RENAME statement to specify 8-character SAS variable names such as access descriptors and view descriptors.
Options That Affect SAS/ACCESS Naming Behavior

To change how SAS handles case-sensitive or nonstandard DBMS table and column names, specify one or more of these options.

PRESERVE_COL_NAMES=YES
This option applies only to creating DBMS tables. When set to YES, it preserves spaces, special characters, and mixed case in DBMS column names. For more information, see the PRESERVE_COL_NAMES= LIBNAME and data set options.

SAP HANA: When this option is set to YES, you are allowed to use SAP HANA reserved words as column names.

PRESERVE_TAB_NAMES=YES
When set to YES, this option preserves blank spaces, special characters, and mixed case in DBMS table names. Specify the PRESERVE_NAMES=YES | NO alias if you plan to specify both the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options in your LIBNAME statement. Using this alias saves you time when you are coding. For more information, see the PRESERVE_TAB_NAMES= LIBNAME option.

SAP HANA: When this option is set to YES, you are allowed to use SAP HANA reserved words as table names.

DQUOTE=ANSI
This PROC SQL option specifies whether PROC SQL treats values within double quotation marks as a character string, or as a column name or table name. When you specify DQUOTE=ANSI, your SAS code can refer to DBMS names that contain characters and spaces that SAS naming conventions do not allow. By specifying DQUOTE=ANSI, you can preserve special characters in table and column names in your SQL statements by enclosing the names in double quotation marks. To preserve table names, you must also specify PRESERVE_TAB_NAMES=YES. To preserve column names when you create a table, you must also specify PRESERVE_COL_NAMES=YES.

VALIDVARNAME=ANY
This global system option can override SAS naming conventions. For more information, see the VALIDVARNAME= system option on page 469.

Examples that use these options are available. The availability of these options and their default settings are DBMS-specific, so see the SAS/ACCESS documentation for your DBMS to learn how the SAS/ACCESS engine for your DBMS processes names.

Naming Behavior When Retrieving DBMS Data

The tables in this section illustrate how SAS/ACCESS processes DBMS names when it retrieves data from a DBMS. This information applies generally to all interfaces. However, in some cases you do not need to specify these options because the option default values are DBMS-specific. For details, see the DBMS-specific reference section for your SAS/ACCESS interface. Available examples illustrate different types of naming actions and defaults.
Table 2.1  DBMS Column Names to SAS Variable Names When Reading DBMS Data

<table>
<thead>
<tr>
<th>DBMS Column Name</th>
<th>Desired SAS Variable Name</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case-sensitive DBMS column name, such as Flight</td>
<td>Case-sensitive SAS variable name, such as Flight</td>
<td>No options are necessary.</td>
</tr>
<tr>
<td>DBMS column name with characters that are not valid in SAS names, such as My$Flight</td>
<td>Case-sensitive SAS variable name where an underscore replaces the invalid characters, such as My_Flight</td>
<td>No options are necessary.</td>
</tr>
<tr>
<td>DBMS column name with characters that are not valid in SAS names, such as My$Flight</td>
<td>Nonstandard, case-sensitive SAS variable name, such as My$Flight</td>
<td>PROC SQL DQUOTE=ANSI or, in a DATA or PROC step, use a SAS name literal such as 'My$Flight'n and VALIDVARNAME=ANY.</td>
</tr>
</tbody>
</table>

Table 2.2  DBMS Table Names to SAS Data Set Names When Reading DBMS Data

<table>
<thead>
<tr>
<th>DBMS Table Name</th>
<th>Desired SAS Data Set Name</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default DBMS table name, such as STAFF</td>
<td>Default SAS data set or member name (uppercase), such as STAFF</td>
<td>PRESERVE_TAB_NAMES=NO</td>
</tr>
<tr>
<td>Case-sensitive DBMS table name, such as Staff</td>
<td>Case-sensitive SAS data set, such as Staff</td>
<td>PRESERVE_TAB_NAMES=YES</td>
</tr>
<tr>
<td>DBMS table name with characters that are not valid in SAS names, such as All$Staff</td>
<td>Nonstandard, case-sensitive SAS data set name, such as All$Staff</td>
<td>PROC SQLDQUOTE=ANSI and PRESERVE_TAB_NAMES=YES or, in a DATA step or PROC, use a SAS name literal such as 'All$Staff'n and PRESERVE_TAB_NAMES=YES</td>
</tr>
</tbody>
</table>

Naming Behavior When Creating DBMS Objects

The tables in this section illustrate how SAS/ACCESS handles variable names when it creates such DBMS objects as tables and views. This information applies generally to all interfaces. However, in some cases you do not need to specify these options because the option default values are DBMS-specific. For details, see the documentation for your DBMS. Available examples illustrate different types of naming actions and defaults.

Table 2.3  SAS Variable Names to DBMS Column Names When Creating Tables

<table>
<thead>
<tr>
<th>SAS Variable Name as Input</th>
<th>Desired DBMS Column Name</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any SAS variable name, such as Miles</td>
<td>Default DBMS column name (normalized to follow the DBMS's naming conventions), such as MILES</td>
<td>PRESERVE_COL_NAMES=NO</td>
</tr>
</tbody>
</table>
## SAS/ACCESS Naming Examples

### Replacing Unsupported Characters

This example creates the view, Myview, from the Oracle table, Mytable.

```sas
proc sql;
connect to oracle (user=myusr1 password=mypwd1);
create view myview as
  select * from connection to oracle
    (select "Amount Budgeted$", "Amount Spent$"
     from mytable);
quit;

proc contents data=myview;
run;
```

In the output that PROC CONTENTS produces, Oracle column names that the SQL View of MYTABLE processed are renamed to different SAS variable names: Amount Budgeted$ becomes Amount_Budgeted_ and Amount Spent$ becomes Amount_Spent_.

### Table 2.4 SAS Data Set Names to DBMS Table Names

<table>
<thead>
<tr>
<th>SAS Data Set Name as Input</th>
<th>Desired DBMS Table Name</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any SAS data set name, such as Payroll</td>
<td>Default DBMS table name (normalized to follow the DBMS's naming conventions), such as PAYROLL</td>
<td>PRESERVE_TAB_NAMES=NO</td>
</tr>
<tr>
<td>Case-sensitive SAS data set name, such as Payroll</td>
<td>Case-sensitive DBMS table name, such as Payroll</td>
<td>PRESERVE_TAB_NAMES=YES</td>
</tr>
<tr>
<td>Case-sensitive SAS data set name with characters that are not valid in a normalized SAS name, such as Payroll-for-QC</td>
<td>Case-sensitive DBMS table name that matches the SAS name, such as Payroll-for-QC</td>
<td>PROC SQL DQUOTE=ANSI and PRESERVE_TAB_NAMES=YES or, in a DATA or PROC step, use a SAS name literal and PRESERVE_TAB_NAMES=YES and VALIDVARNAME=ANY</td>
</tr>
</tbody>
</table>

### Table 2.5 SAS Variable Name as Input

<table>
<thead>
<tr>
<th>SAS Variable Name as Input</th>
<th>Desired DBMS Column Name</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>A case-sensitive SAS variable name, such as Miles</td>
<td>Case-sensitive DBMS column name, such as Miles</td>
<td>PRESERVE_COL_NAMES=YES</td>
</tr>
<tr>
<td>A SAS variable name with characters that are not valid in a normalized SAS name, such as Miles-to-Go</td>
<td>Case-sensitive DBMS column name that matches the SAS name, such as Miles-to-Go</td>
<td>PROC SQL DQUOTE=ANSI and PRESERVE_COL_NAMES=YES or, in a DATA or PROC step, use a SAS name literal and PRESERVE_COL_NAMES=YES and VALIDVARNAME=ANY</td>
</tr>
</tbody>
</table>

---

**SAS/ACCESS Naming Examples 19**
Preserving Column Names

In this example, the Oracle table, PAYROLL, creates a new Oracle table, PAY1, which it then prints. You can use both PRESERVE_COL_NAMES=YES and the PROC SQL DQUOTE=ANSI options to preserve the case and nonstandard characters in the column names. You do not need to quote the column aliases to preserve mixed case. You need only double quotation marks when the column name has nonstandard characters or blanks.

By default, most SAS/ACCESS interfaces use DBMS-specific rules to set the case of table and column names. So, even though the new pay1 Oracle table name in this example is created in lowercase, Oracle stores the name in uppercase as PAY1. To store the table name as "pay1", set PRESERVE_TAB_NAMES=NO.

By default, most SAS/ACCESS interfaces use DBMS-specific rules to set the case of table and column names. So, even though the new pay1 Oracle table name in this example is created in lowercase, Oracle stores the name in uppercase as PAY1. To store the table name as "pay1", set PRESERVE_TAB_NAMES=NO.

By default, most SAS/ACCESS interfaces use DBMS-specific rules to set the case of table and column names. So, even though the new pay1 Oracle table name in this example is created in lowercase, Oracle stores the name in uppercase as PAY1. To store the table name as "pay1", set PRESERVE_TAB_NAMES=NO.

options linesize=120 pagesize=60 nodate;
libname mydblib oracle user=myusr1 password=mypwd1 path='mysrv1' schema=hrdept preserve_col_names=yes;
proc sql dquote=ansi;
create table mydblib.pay1 as
  select idnum as "ID #", sex, jobcode, salary,
       birth as BirthDate, hired as HiredDate
  from mydblib.payroll
order by birth;
title "Payroll Table with Revised Column Names";
select * from mydblib.pay1;
quit;

SAS recognizes the JOBCODE, SEX, and SALARY column names, regardless of how you specify them in your SAS code: as lowercase, mixed case, or uppercase. SEX, JOBCODE, and SALARY columns in the PAYROLL Oracle table were created in uppercase. So they retain this case in the new table unless you rename them. Here is partial output from the example.

Output 2.2  DBMS Table Created with Nonstandard and Standard Column Names

<table>
<thead>
<tr>
<th>ID #</th>
<th>SEX</th>
<th>JOBCODE</th>
<th>SALARY</th>
<th>BirthDate</th>
<th>HiredDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1118</td>
<td>M</td>
<td>PT3</td>
<td>11379</td>
<td>16JAN1944:00:00:00</td>
<td>18DEC1980:00:00:00</td>
</tr>
<tr>
<td>1065</td>
<td>M</td>
<td>ME2</td>
<td>35090</td>
<td>26JAN1944:00:00:00</td>
<td>07JAN1987:00:00:00</td>
</tr>
<tr>
<td>1409</td>
<td>M</td>
<td>ME3</td>
<td>41551</td>
<td>19APR1950:00:00:00</td>
<td>22OCT1981:00:00:00</td>
</tr>
<tr>
<td>1401</td>
<td>M</td>
<td>TA3</td>
<td>38822</td>
<td>13DEC1950:00:00:00</td>
<td>17NOV1985:00:00:00</td>
</tr>
<tr>
<td>1890</td>
<td>M</td>
<td>PT2</td>
<td>91908</td>
<td>20JUL1951:00:00:00</td>
<td>25NOV1979:00:00:00</td>
</tr>
</tbody>
</table>

Preserving Table Names

This example uses PROC PRINT to print the DBMS table PAYROLL. The DBMS table was created in uppercase. Because PRESERVE_TAB_NAMES=YES, you must specify
the table name in uppercase. (If you set the PRESERVE_TAB_NAMES=NO, you can specify the DBMS table name in lowercase.) Partial output follows the example.

```
options nodate linesize=64;
libname mydblib oracle user=myusr1 password=mypwd1
    path='mysrv1' preserve_tab_names=yes;
```

```
proc print data=mydblib.PAYROLL;
    title 'PAYROLL Table';
run;
```

**Output 2.3**  
**DBMS Table with a Case-Sensitive Name**

<table>
<thead>
<tr>
<th>Obs</th>
<th>IDNUM</th>
<th>SEX</th>
<th>JOBCODE</th>
<th>SALARY</th>
<th>BIRTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1919</td>
<td>M</td>
<td>TA2</td>
<td>34376</td>
<td>12SEP1960:00:00:00</td>
</tr>
<tr>
<td>2</td>
<td>1653</td>
<td>F</td>
<td>ME2</td>
<td>35108</td>
<td>15OCT1964:00:00:00</td>
</tr>
<tr>
<td>3</td>
<td>1400</td>
<td>M</td>
<td>ME1</td>
<td>29769</td>
<td>05NOV1967:00:00:00</td>
</tr>
<tr>
<td>4</td>
<td>1350</td>
<td>F</td>
<td>FA3</td>
<td>32886</td>
<td>31AUG1965:00:00:00</td>
</tr>
<tr>
<td>5</td>
<td>1401</td>
<td>M</td>
<td>TA3</td>
<td>38822</td>
<td>13DEC1950:00:00:00</td>
</tr>
</tbody>
</table>

This next example submits a SAS/ACCESS LIBNAME statement. It then opens the SAS Explorer window, which lists the Oracle tables and views that the MYDBLIB libref references. The 16 members are listed, and all member names are in the case (initial capitalization) that the Explorer window sets. Table names are capitalized due to the PRESERVE_TAB_NAMES=NO default.

```
libname mydblib oracle user=myusr1 pass=mypwd1;
```

**Figure 2.1**  
**SAS Explorer Window Listing DBMS Objects**

If you submit a SAS/ACCESS LIBNAME statement with PRESERVE_TAB_NAMES=YES and then open the SAS Explorer window, you see a different listing of the Oracle tables and views that the MYDBLIB libref references.

```
libname mydblib oracle user=myusr1 password=mypwd1
    preserve_tab_names=yes;
```
The 18 members are listed, including one in lowercase and one with a name that is separated by a blank space. Because PRESERVE_TAB_NAMES=YES, SAS displays the tables names in the exact case in which they were created.

**Using DQUOTE=ANSI**

This example creates a DBMS table with a blank space in its name. It uses double quotation marks to specify the table name, International Delays. You can also set both of the preserve names LIBNAME options by using the alias PRESERVE_NAMES=.

Because PRESERVE_NAMES=YES, the schema airport is now case sensitive for Oracle.

```sas
options linesize=64 nodate;
libname mydblib oracle user=myusr1 password=mypwd1 path='airdata'
    schema=airport preserve_names=yes;
proc sql dquote=ansi;
create table mydblib."International Delays" as
    select int.flight as "FLIGHT NUMBER", int.dates,
        del.orig as ORIGIN,
        int.dest as DESTINATION, del.delay
    from mydblib.INTERNAT as int,
        mydblib.DELAY as del
    where int.dest=del.dest and int.dest='LON';
quitar;
proc sql dquote=ansi outobs=10;
    title "International Delays";
    select * from mydblib."International Delays";
```

You can use single quotation marks to specify the data value for London (int.dest='LON') in the WHERE clause. Because of the preserve name LIBNAME options, using double quotation marks would cause SAS to interpret this data value as a column name.
Using a label to change the name of a DBMS column name changes only the output. Enclose the label in single quotation marks. Because this column name and the table name (International Delays) each contain a space in their names, you must enclose the names in double quotation marks. Partial output follows the example.

```
options linesize=64 nodate;
libname mydblib oracle user=myusr1 password=mypwd1 path='airdata' schema=airport preserve_names=yes;
proc sql dquote=ansi outobs=5;
title "Query from International Delays";
select "FLIGHT NUMBER" label='Flight_Number', dates, delay
from mydblib."International Delays";
```

You can preserve special characters by specifying DQUOTE=ANSI and using double quotation marks around the SAS names in your SELECT statement.

```
proc sql dquote=ansi;
    connect to oracle {user=myusr1 password=mypwd1};
    create view myview as
        select "Amount Budgeted$", "Amount Spent$"
        from connection to oracle
            (select "Amount Budgeted$", "Amount Spent$"
                from mytable);
    quit;
proc contents data=myview;
```
Using Name Literals

This example creates a table using name literals. To use name literals, you must specify the SAS option VALIDVARNAME=ANY. Use PROC SQL to print the new DBMS table because name literals work only with PROC SQL and the DATA step. 

\[ \text{PROC SQL;} \]

\[ \text{title "Sample Table";} \]

\[ \text{select * from mydblib.'Sample Table';} \]

**Output 2.6** DBMS Table to Test Column Names

<table>
<thead>
<tr>
<th>Sample Table</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpID#</td>
<td>Lname</td>
<td>Salary in $</td>
</tr>
<tr>
<td>12345</td>
<td>Chen</td>
<td>63000</td>
</tr>
</tbody>
</table>

Using DBMS Data to Create a DBMS Table

This example uses PROC SQL to create a DBMS table that is based on data from other DBMS tables. To preserve the case sensitivity of the aliased column names, use \text{PRESERVE\_COL\_NAMES=YES}. Partial output follows the example.

\[ \text{libname mydblib oracle user=myusr1 password=mypwd1 path='hrdata99' schema=personnel preserve\_col\_names=yes;} \]

\[ \text{proc sql;} \]

\[ \text{create table mydblib.gtforty as} \]

\[ \text{select lname as LAST\_NAME,} \]

\[ \text{fname as FIRST\_NAME,} \]

\[ \text{salary as ANNUAL\_SALARY} \]

\[ \text{from mydblib.staff a,} \]

\[ \text{mydblib.payroll b} \]

\[ \text{where (a.idnum eq b.idnum) and} \]

\[ \text{(salary gt 40000)} \]

\[ \text{order by lname;} \]
proc print noobs;
  title 'Employees with Salaries Greater Than $40,000';
run;

Output 2.7 Updating DBMS Data

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANADYGA</td>
<td>JUSTIN</td>
<td>88606</td>
</tr>
<tr>
<td>BAREFOOT</td>
<td>JOSEPH</td>
<td>43025</td>
</tr>
<tr>
<td>BRADY</td>
<td>CHRISTINE</td>
<td>68767</td>
</tr>
<tr>
<td>BRANCACCIO</td>
<td>JOSEPH</td>
<td>66517</td>
</tr>
<tr>
<td>CARTER-COHEN</td>
<td>KAREN</td>
<td>40260</td>
</tr>
<tr>
<td>CASTON</td>
<td>FRANKLIN</td>
<td>41690</td>
</tr>
<tr>
<td>COHEN</td>
<td>LEE</td>
<td>91376</td>
</tr>
<tr>
<td>FERNANDEZ</td>
<td>KATRINA</td>
<td>51081</td>
</tr>
</tbody>
</table>

Using a SAS Data Set to Create a DBMS Table

This example uses a SAS DATA step to create a DBMS table, College-Hires-1999, from a temporary SAS data set that has case-sensitive names. It creates the temporary data set and defines the LIBNAME statement. Because it uses a DATA step to create the DBMS table, it must specify the table name as a name literal and specify the PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options. In this case, it performs these actions by using the alias PRESERVE_NAMES=.

options validvarname=any nodate;

data College_Hires_1999;
  input IDnum $4. +3 Lastname $11. +2
    Firstname $10. +2 City $15. +2
    State $2.;
  datalines;
  3413 Schwartz Robert New Canaan CT
  3523 Janssen Heike Stamford CT
  3565 Gomez Luis Darien CT
;
libname mydblib oracle user=myusr1 password=mypwd1
  path='hrdata99' schema=hrdept preserve_names=yes;
data mydblib.'College-Hires-1999'n;
set College_Hires_1999;
proc print;
  title 'College Hires in 1999';
run;
### Output 2.8  DBMS Table with Case-Sensitive Table and Column Names

<table>
<thead>
<tr>
<th>Obs</th>
<th>IDnum</th>
<th>Lastname</th>
<th>Firstname</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3413</td>
<td>Schwartz</td>
<td>Robert</td>
<td>New Canaan</td>
<td>CT</td>
</tr>
<tr>
<td>2</td>
<td>3523</td>
<td>Janssen</td>
<td>Heike</td>
<td>Stamford</td>
<td>CT</td>
</tr>
<tr>
<td>3</td>
<td>3565</td>
<td>Gomez</td>
<td>Luis</td>
<td>Darien</td>
<td>CT</td>
</tr>
</tbody>
</table>
Chapter 3
Data Integrity and Security

Introduction to Data Integrity and Security

Here is what this section covers.

- a brief description of DBMS security issues
- measures that you can take from within SAS to help protect DBMS data from accidental update or deletion
- how SAS handles null values to help you achieve consistent results

DBMS Security

Privileges

Database administrators control who has privileges to access or update DBMS objects. They also control who can create objects, and object creators control who can access the objects. Users cannot use DBMS facilities to access DBMS objects through SAS/ACCESS software unless they have the appropriate DBMS privileges or authority.
on those objects. You can grant privileges on the DBMS side by using the SQL pass-through facility to EXECUTE an SQL statement or by issuing a GRANT statement from the DBLOAD procedure.

You should give users only the privileges on the DBMS that they must have. Privileges are granted on whole tables or views. You must explicitly grant user privileges on the DBMS tables or views that underlie a view so that users can use that view.

For more information about ensuring security on the DBMS side of the interface, see your DBMS documentation.

**Triggers**

If your DBMS supports triggers, you can use them to enforce security authorizations or business-specific security considerations. When and how triggers are executed is determined by when the SQL statement is executed and how often the trigger is executed. Triggers can be executed before an SQL statement is executed, after an SQL statement is executed, or for each row of an SQL statement. Also, triggers can be defined for DELETE, INSERT, and UPDATE statement execution.

Enabling triggers can provide more specific security for Delete, Insert, and Update operations. SAS/ACCESS abides by all constraints and actions that are specified by a trigger. For more information, see the documentation for your DBMS.

**SAS Security**

**Securing Data**

SAS preserves the data security provided by your DBMS and operating system; SAS/ACCESS does not override the security of your DBMS. To secure DBMS data from accidental update or deletion, from the SAS side of the interface, you can take steps like these.

- Specify the SAS/ACCESS DBPROMPT=LIBNAME option to avoid saving connection information in your code.
- Create SQL views and protecting them from unauthorized access by applying passwords.

These and other approaches are discussed in detail in subsequent sections.

**Assigning SAS Passwords**

By using SAS passwords, you can protect SQL views, SAS data sets, and descriptor files from unauthorized access. The following table summarizes the levels of protection that SAS passwords provide. Note that you can assign multiple levels of protection.
Table 3.1 Password Protection Levels and Their Effects

<table>
<thead>
<tr>
<th>File Type</th>
<th>READ=</th>
<th>WRITE=</th>
<th>ALTER=</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC SQL view of DBMS data</td>
<td>Protects the underlying data from being read or updated through the view; does not protect against replacement of the view</td>
<td>Protects the underlying data from being updated through the view; does not protect against replacement of the view</td>
<td>Protects the view from being modified, deleted, or replaced</td>
</tr>
<tr>
<td>Access descriptor</td>
<td>No effect on descriptor</td>
<td>No effect on descriptor</td>
<td>Protects the descriptor from being read or edited</td>
</tr>
<tr>
<td>View descriptor</td>
<td>Protects the underlying data from being read or updated through the view</td>
<td>Protects the underlying data from being updated through the view</td>
<td>Protects the descriptor from being read or edited</td>
</tr>
</tbody>
</table>

You can use these methods to assign, change, or delete a SAS password:

- the global SETPASSWORD command, which opens a dialog box
- the DATASETS procedure's MODIFY statement

Here is the syntax for using PROC DATASETS to assign a password to an access descriptor, a view descriptor, or a SAS data file.

```
PROC DATASETS LIBRARY=libref MEMTYPE=member-type;
MODIFY member-name (password-level = password-modification);
RUN;
```

The `password-level` argument can have one or more of these values: READ=, WRITE=, ALTER=, or PW=. PW= assigns Read, Write, and Alter privileges to a descriptor or data file. The `password-modification` argument enables you to assign a new password or to change or delete an existing password. For example, this PROC DATASETS statement assigns the password MONEY with the ALTER level of protection to the access descriptor ADLIB.SALARIES:

```
proc datasets library=adlib memtype=access;
    modify salaries (alter=money);
run;
```

In this case, users are prompted for the password whenever they try to browse or update the access descriptor or try to create view descriptors that are based on ADLIB.SALARIES.

In the next example, the PROC DATASETS statement assigns the passwords MYPW and MYDEPT with READ and ALTER levels of protection to the view descriptor VLIB.JOBC204:

```
proc datasets library=vlib memtype=view;
    modify jobc204 (read=mypw alter=mydept);
run;
```

In this case, users are prompted for the SAS password when they try to read the DBMS data or try to browse or update the view descriptor VLIB.JOBC204. You need both levels to protect the data and descriptor from being read. However, a user could still...
update the data that VLIB.JOBC204 accesses (for example, using a PROC SQL UPDATE. Assign a WRITE level of protection to prevent data updates).

When you assign multiple levels of passwords, use a different password for each level to ensure that you grant only the access privileges that you intend.

To delete a password, put a slash after the password:

```
proc datasets library=vlib memtype=view;
    modify jobc204 (read=mypw/ alter=mydept/);
run;
```

---

**Protecting Connection Information**

In addition to directly controlling access to data, you can protect the data indirectly by protecting the connection information that SAS/ACCESS uses to reach the DBMS. Generally, you can achieve this by not saving connection information in your code.

One way to protect connection information is by storing user name, password, and other connection options in a local environment variable. Access to the DBMS is denied unless the correct user and password information is stored in a local environment variable. See the documentation for your DBMS to determine whether this alternative is supported.

Another way to protect connection information is by requiring users to manually enter it at connection time. When you specify `DBPROMPT=YES` in a SAS/ACCESS LIBNAME statement, each user has to provide DBMS connection information in a dynamic, interactive manner. The statement below opens a dialog box to prompt the user to enter connection information, such as a user name and password:

```
libname myoralib oracle dbprompt=yes defer=no;
```

The dialog box that appears contains the DBMS connection options that are valid for the SAS/ACCESS engine that is being used; in this case, Oracle.

Using the `DBPROMPT=` option in the LIBNAME statement offers several advantages. DBMS account passwords are protected because they do not need to be stored in a SAS program or descriptor file. Also, when a password or user name changes, the SAS program does not need to be modified. Another advantage is that the same SAS program can be used by any valid user name and password combination that is specified during execution. You can also use connection options in this interactive manner when you want to run a program on a production server instead of testing a server without modifying your code. By using the prompt window, the new server name can be specified dynamically.

The `DBPROMPT=` option is not available in SAS/ACCESS Interface to DB2 under z/OS.

---

**Extracting DBMS Data to a SAS Data Set**

If you are the owner of a DBMS table and do not want anyone else to read the data, you can extract the data (or a subset of the data) and not distribute information about either the access descriptor or view descriptor.

You might need to take additional steps to restrict LIBNAME or pass-through access to the extracted data set.

If you extract data from a view with an assigned SAS password, the new SAS data file is automatically assigned to the same password. If a view does not have a password, you
can assign a password to the extracted SAS data file by using the MODIFY statement in the DATASETS procedure. For more information, see the *Base SAS Procedures Guide*.

**Defining Views and Schemas**

If you want to provide access to some but not all fields in a DBMS table, create a SAS view that prohibits access to the sensitive data. To restrict access to some columns, specify that those columns be dropped. Columns that are dropped from views do not affect the underlying DBMS table and can be reselected for later use.

Some SAS/ACCESS engines support LIBNAME options that restrict or qualify the scope, or schema, of the tables in the libref. For example, the DB2 engine supports the AUTHID= and LOCATION= options, and the Oracle engine supports the SCHEMA= and DBLINK= options. See the SAS/ACCESS documentation for your DBMS to determine which options are available to you.

This example uses SAS/ACCESS Interface to Oracle.

```sas
libname myoralib oracle user=myusr1 password=mypwd1 path='mysrv1' schema=testgroup;
```

```sas
proc datasets lib=myoralib;
run;
```

In this example, the MYORALIB libref is associated with the Oracle schema named TESTGROUP. The DATASETS procedure lists only the tables and views that are accessible to the TESTGROUP schema. Any reference to a table that uses the MYORALIB libref is passed to the Oracle server as a qualified table name. For example, if the SAS program reads a table by specifying the SAS data set MYSRV1.TESTTABLE, the SAS/ACCESS engine passes this query to the server.

```sas
select * from "testgroup.testtable"
```

**Controlling DBMS Connections**

Because the overhead of executing a connection to a DBMS server can be resource-intensive, SAS/ACCESS supports the CONNECTION= and DEFER= options to control when a DBMS connection is made, and how many connections are executed within the context of your SAS/ACCESS application. For most SAS/ACCESS engines, a connection to a DBMS begins one transaction, or work unit, and all statements issued in the connection execute within the context of the active transaction.

The CONNECTION= LIBNAME option enables you to specify how many connections are executed when the library is used and which operations on tables are shared within a connection. By default, the value is CONNECTION=SHAREDREAD, which means that a SAS/ACCESS engine executes a shared read DBMS connection when the library is assigned. Every time a table in the library is read, the read-only connection is used. However, if an application attempts to update data using the libref, a separate connection is issued, and the update occurs in the new connection. As a result, there is one connection for read-only transactions and a separate connection for each update transaction.

In the example below, the SAS/ACCESS engine issues a connection to the DBMS when the libref is assigned. The PRINT procedure reads the table by using the first connection. When the PROC SQL updates the table, the update is performed with a second connection to the DBMS.

```sas
libname myoralib oracle user=myusr1 password=mypwd1
```
path='mysrv1';

proc print data=myoralib.mytable;
run;

proc sql;
  update myoralib.mytable set acctnum=123
  where acctnum=456;
quit;

This example uses SAS/ACCESS Interface to DB2 under z/OS. The LIBNAME statement executes a connection by way of the DB2 Call Attach Facility to the DB2 DBMS server.

libname mydb2lib db2 authid=myusr1;

To assign more than one SAS libref to your DBMS server when you do not plan to update the DBMS tables, SAS/ACCESS lets you optimize how the engine makes connections. Your SAS librefs can share a single read-only connection to the DBMS if you use the CONNECTION=GLOBALREAD option. This example shows how to use the CONNECTION= option with the ACCESS= option to control your connection and to specify read-only data access.

libname mydblib1 db2 authid=myusr1 connection=globalread access=readonly;

If you do not want the connection to occur when the library is assigned, you can delay the connection to the DBMS by using the DEFER= option. When you specify DEFER=YES in the LIBNAME statement, the SAS/ACCESS engine connects to the DBMS the first time a DBMS object is referenced in a SAS program:

libname mydb2lib db2 authid=myusr1 defer=yes;

Note: If you use DEFER=YES to assign librefs to your DBMS tables and views in an AUTOEXEC program, the processing of the AUTOEXEC file is faster. The processing is faster because the connections to the DBMS are not made every time SAS is invoked.

**Locking, Transactions, and Currency Control**

SAS/ACCESS provides options so that you can control some of the row, page, or table locking operations that the DBMS and SAS/ACCESS engine perform as your programs are executed. For example, by default, the SAS/ACCESS Oracle engine does not lock any data when it reads rows from Oracle tables. However, you can override this behavior by using the locking options that SAS/ACCESS Interface to Oracle supports.

To lock the data pages of a table while SAS is reading the data to prevent other processes from updating the table, use the READLOCK_TYPE= option, as shown in this example.

libname myoralib oracle user=myusr1 pass=mypwd1
  path='mysrv1' readlock_type=table;

data work.mydata;
  set myoralib.mytable(where=(colnum > 123));
run;

Here the SAS/ACCESS Oracle engine obtains a TABLE SHARE lock on the table so that other processes cannot update the data while your SAS program reads it.
In this next example, Oracle acquires row-level locks on rows read for update in the tables in the libref.

```plaintext
libname myoralib oracle user=myusr1 password=mypwd1 path='mysrv1' updatelock_type=row;
```

Each SAS/ACCESS interface supports specific options. See the DBMS-specific reference section for your SAS/ACCESS interface to determine which options it supports.

### Customizing DBMS Connect and Disconnect Exits

To specify DBMS commands or stored procedures to run immediately after a DBMS connection or before a DBMS disconnect, use the `DBCONINIT=` and `DBCONTERM=` `LIBNAME` `LIBNAME` options. Here is an example.

```plaintext
libname myoralib oracle user=myusr1 password=mypwd1 path='mysrv1' dbconinit="EXEC MY_PROCEDURE";
```

```plaintext
proc sql;
  update myoralib.mytable set acctnum=123 where acctnum=567;
quit;
```

When the libref is assigned, the SAS/ACCESS engine connects to the DBMS and passes a command to the DBMS to execute the stored procedure `MY_PROCEDURE`. By default, a new connection to the DBMS is made for every table that is opened for updating. Therefore, `MY_PROCEDURE` is executed a second time after a connection is made to update the table `MYTABLE`.

To execute a DBMS command or stored procedure only after the first connection in a library assignment, you can use the `DBLIBINIT=` option. Similarly, you can use the `DBLIBTERM=` `LIBNAME` option to specify a command to run before the disconnection of only the first library connection. Here is an example.

```plaintext
libname myoralib oracle user=myusr1 password=mypwd1 dblibinit="EXEC MY_INIT" dblibterm="EXEC MY_TERM";
```

### Potential Result Set Differences When Processing Null Data

When your data contains null values or when internal processing generates intermediate data sets that contain null values, you might receive different results depending on whether the processing is done by SAS or by the DBMS. Although in many cases this does not present a problem, it is important to understand how these differences occur.

Most relational database systems have a special value called null, which means an absence of information and is analogous to a SAS missing value. SAS/ACCESS translates SAS missing values to DBMS null values when creating DBMS tables from within SAS. Conversely, SAS/ACCESS translates DBMS null values to SAS missing values when reading DBMS data into SAS.

However, there is an important difference in the behavior of DBMS null values and SAS missing values.
A DBMS null value is interpreted as the absence of data, so you cannot sort a DBMS null value or evaluate it with standard comparison operators.

A SAS missing value is interpreted as its internal floating-point representation because SAS supports 28 missing values (where a period (.) is the most common missing value). Because SAS supports multiple missing values, you can sort a SAS missing value and evaluate it with standard comparison operators.

This means that SAS and the DBMS interpret null values differently, which has significant implications when SAS/ACCESS passes queries to a DBMS for processing. This can be an issue in these situations:

- when filtering data (for example, in a WHERE clause, a HAVING clause, or an outer join ON clause). SAS interprets null values as missing; many DBMSs exclude null values from consideration. For example, if you have null values in a DBMS column that is used in a WHERE clause, your results might differ depending on whether the WHERE clause is processed in SAS or is passed to the DBMS for processing. This is because the DBMS removes null values from consideration in a WHERE clause, but SAS does not.

- when using certain functions. For example, if you use the MIN aggregate function on a DBMS column that contains null values, the DBMS does not consider the null values, but SAS interprets the null values as missing. This interpretation affects the result.

- when submitting outer joins where internal processing generates nulls for intermediate result sets.

- when sorting data. SAS sorts null values low; most DBMSs sort null values high. For more information, see “Sorting DBMS Data” on page 39.

For example, create a simple data set that consists of one observation and one variable.

```
libname myoralib oracle user=myusr1 password=mypwd1;
data myoralib.table;
x=.;       /*create a missing value */
run;
```

Then print the data set using a WHERE clause, which SAS/ACCESS passes to the DBMS for processing.

```
proc print data=myoralib.table;
  where x<0;
run;
```

The log indicates that the WHERE clause selected no observations. Oracle interprets the missing value as the absence of data and does not evaluate it with the less-than (<) comparison operator.

When there is the potential for inconsistency, consider using one of these strategies.

- Use the `DIRECT_SQL=` LIBNAME option to control whether SAS or the DBMS handles processing.
- Use the `SQL pass-through facility` to ensure that the DBMS handles processing.
- Add the `is not null` expression to WHERE clauses and ON clauses to ensure that you obtain the same result regardless of whether SAS or the DBMS does the processing.

Use the `NULLCHAR=` data set option to specify how the DBMS interprets missing SAS character values when updating DBMS data or inserting rows into a DBMS table.

You can use the first of these strategies to force SAS to process the data in this example.
direct_sql=nowhere; /* forces SAS to process WHERE clauses */
data myoralib.table;
x=.; /* create a missing value */
run;

You can then print the data set using a WHERE clause
proc print data=myoralib.table;
   where x<0;
run;

This time the log indicates that one observation was read from the data set because SAS evaluates the missing value as satisfying the less-than-zero condition in the WHERE clause.
Chapter 4
Performance Considerations

Increasing Throughput of the SAS Server

When you start SAS as a server that responds to multiple clients, you can use the DBSRVTP= system option to improve the performance of the clients. This option tells the SAS server whether to put a hold (block) on the originating client while making performance-critical calls to the database. By holding or blocking the originating client, the SAS/ACCESS server remains available for other clients; they do not need to wait for the originating client to complete its call to the database.

Limiting Retrieval

Row and Column Selection

Limiting the number of rows that the DBMS returns to SAS is an extremely important performance consideration. The less data that the SAS job requests, the faster the job runs.

Wherever possible, specify selection criteria that limit the number of rows that the DBMS returns to SAS. Use the SAS WHERE clause to retrieve a subset of the DBMS data.
If you are interested in only the first few rows of a table, consider adding the OBS= option. SAS passes this option to the DBMS to limit the number of rows to transmit across the network, which can significantly improve performance against larger tables. To do this if you are using SAS Enterprise Guide, select View ➪ Explorer, select the table that you want from the list of tables, and select the member that you want to see the contents of the table.

Likewise, select only the DBMS columns that your program needs. Selecting unnecessary columns slows your job.

**The KEEP= and DROP= Options**

Just as with a SAS data set you can use the DROP= and KEEP= data set options to prevent retrieving unneeded columns from your DBMS table. In this example, the KEEP= data set option causes the SAS/ACCESS engine to select only the SALARY and DEPT columns when it reads the MYDBLIB.EMPLOYEES table.

```sas
libname mydblib db2 user=testid password=testpass database=testdb;
proc print data (keep=salary dept);
  where dept='ACC024';
quit;
```

The DBMS generates SQL that is similar to this:

```
SELECT "SALARY", "DEPT" FROM EMPLOYEES
  WHERE(DEPT="ACC024")
```

Without the KEEP option, the DBMS processes SQL that is similar to this code:

```
SELECT * FROM EMPLOYEES  WHERE(DEPT="ACC024")
```

This results in all columns from the EMPLOYEES table being read in to SAS. The DROP= data set option is a parallel option that specifies columns to omit from the output table. Keep in mind that the DROP= and KEEP= data set options are not interchangeable with the DROP and KEEP statements. Use of the DROP and KEEP statements when selecting data from a DBMS can result in retrieval of all columns into SAS, which can seriously impact performance.

For example, this code results in all columns from the EMPLOYEES table being retrieved into SAS. When creating the output data set, the KEEP statement is applied.

```sas
libname mydblib db2 user=testid password=testpass database=testdb;

data temp;
  set mydblib.employees;
  keep salary;
run;
```

Here is how you can use the KEEP= data set option to retrieve only the SALARY column.

```sas
data temp;
  set mydblib.employees(keep=salary);
run;
```
Repeatedly Accessing Data

**CAUTION:**
If you need to access the most current DBMS data, access it directly from the database every time. Do not follow the extraction suggestions in this section.

It is sometimes more efficient to extract (copy) DBMS data to a SAS data file than to repeatedly read the data by using a SAS view. SAS data files are organized to provide optimal performance with PROC and DATA steps. Programs that use SAS data files are often more efficient than SAS programs that read DBMS data directly.

Consider extracting data when you work with a large DBMS table and plan to use the same DBMS data in several procedures or DATA steps during the same SAS session.

You can extract DBMS data to a SAS data file by using the OUT= option, a DATA step, or ACCESS procedures.

---

Sorting DBMS Data

Sorting DBMS data can be resource-intensive—whether you use the SORT procedure, a BY statement, or an ORDER BY clause on a DBMS data source or in the SQL procedure SELECT statement. Sort data only when it is needed for your program.

Here are guidelines for sorting data.

- If you specify a BY statement in a DATA or PROC step that references a DBMS data source, it is recommended for performance reasons that you associate the BY variable with an indexed DBMS column. If you reference DBMS data in a SAS program and the program includes a BY statement for a variable that corresponds to a column in the DBMS table, the SAS/ACCESS LIBNAME engine automatically generates an ORDER BY clause for that variable. The ORDER BY clause causes the DBMS to sort the data before the DATA or PROC step uses the data in a SAS program. If the DBMS table is very large, this sorting can adversely affect your performance. Use a BY variable that is based on an indexed DBMS column in order to reduce this negative impact.

- The outermost BY or ORDER BY clause overrides any embedded BY or ORDER BY clauses. This includes those specified by the DBCONDITION= option, in a WHERE clause, and in the selection criteria in a view descriptor. In the following example, the EXEC_EMPLOYEES data set includes a BY statement that sorts data by the SENIORITY variable. However, when that data set is used in the following PROC SQL query, the data is ordered by the SALARY column, not by SENIORITY.

```sql
libname mydblib oracle user=myusr1 password=mypwd1;
data exec_employees;
  set mydblib.staff (keep=lname fname idnum);
  by seniority;
  where salary >= 150000;
run;
proc sql;
select * from exec_employees
  order by salary;
```
• Do not use PROC SORT to sort data from SAS back into the DBMS because this impedes performance and has no effect on the order of the data.

• Do not use the SORTSEQ= system option because this option has no effect on the sort order of the data. Whenever possible, SAS allows sorting to be performed by the DBMS to improve performance. The values for the SORTSEQ= system option apply to processing that is performed by SAS. Therefore, the option has no impact when data is sorted by the DBMS.

• The database does not guarantee sort stability when you use PROC SORT. Sort stability means that the ordering of the observations in the BY statement is exactly the same every time the sort is run against static data. If you absolutely require sort stability, you must place your database data into a SAS data set and use PROC SORT.

• When you use PROC SORT, be aware that the sort rules for SAS and for your DBMS might be different. Use the Base SAS system option SORTPGM to specify which rules (host, SAS, or DBMS) are applied:

  SORTPGM=BEST
  sorts data according to the DBMS sort rules, the host sort rules, and the SAS sort rules. (Sorting uses the first available and pertinent sorting algorithm in this list.)
  This is the default.

  SORTPGM=HOST
  sorts data according to host rules and then SAS rules. (Sorting uses the first available and pertinent sorting algorithm in this list.)

  SORTPGM=SAS
  sorts data by SAS rules.

---

**Temporary Table Support for SAS/ACCESS**

**Overview**

DBMS temporary table support in SAS consists of the ability to retain DBMS temporary tables from one SAS step to the next. This ability is a result of establishing a SAS connection to the DBMS that persists across multiple SAS procedures and DATA steps. Temporary table support is available for these DBMSs.

**Table 4.1  DBMS-Specific Temporary Table Support**

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Temporary Table Support</th>
<th>DBMSTEMP= LIBNAME Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aster</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>DB2</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Greenplum</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Hadoop</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Impala</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
The value of DBMS temporary table support in SAS is increased performance potential. By pushing the processing to the DBMS in certain situations, you can achieve an overall performance gain. The processes in this section provide a general outline of how to use DBMS temporary tables.

**General Temporary Table Use**

Follow these steps to use temporary tables on the DBMS.

1. Establish a global connection to the DBMS that persists across SAS procedure and DATA step boundaries.
2. Create a DBMS temporary table and load it with data.
3. Use the DBMS temporary table with SAS.

Closing the global connection causes the DBMS temporary table to close as well.

**Pushing Heterogeneous Joins**

Follow these steps to push heterogeneous joins to the DBMS.

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Temporary Table Support</th>
<th>DBMSTEMP= LIBNAME Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informix</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>MySQL</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Netezza</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>ODBC</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OLE DB</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Oracle</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SAP HANA</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Sybase</td>
<td>yes</td>
<td>no*</td>
</tr>
<tr>
<td>Sybase IQ</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Teradata</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Vertica</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

* The Sybase engine uses the same special table-name syntax as the native database to create a temporary table.
1. Establish a global connection to the DBMS that persists across SAS procedure and DATA step boundaries.
2. Create a DBMS temporary table and load it with data.
3. Perform a join on the DBMS using the DBMS temporary and DBMS permanent tables.
4. Process the result of the join with SAS.

**Pushing Updates**

To push updates (process transactions) to the DBMS:

1. Establish a global connection to the DBMS that persists across SAS procedure and DATA step boundaries.
2. Create a DBMS temporary table and load it with data.
3. Issue SQL that uses values in the temporary table to process against the production table.
4. Process the updated DBMS tables with SAS.

Although these processing scenarios are purposely generic, they apply to each DBMS that supports temporary tables. For details, see the “DBMSTEMP= LIBNAME Option” on page 143.
Chapter 5
Optimizing Your SQL Usage

Overview: Optimizing Your SQL Usage
SAS/ACCESS takes advantage of DBMS capabilities by passing certain SQL operations to the DBMS whenever possible. This can reduce data movement, which can improve performance. The performance impact can be significant when you access large DBMS tables and the SQL that is passed to the DBMS subsets the table to reduce the number of rows. SAS/ACCESS sends operations to the DBMS for processing in these situations.

- When you use the SQL pass-through facility, you submit DBMS-specific SQL statements that are sent directly to the DBMS for execution. For example, when you submit Transact-SQL statements to be passed to a Sybase database.
- When SAS/ACCESS can translate the operations into the SQL of the DBMS. When you use the SAS/ACCESS LIBNAME statement, you submit SAS statements that SAS/ACCESS can often translate into the SQL of the DBMS and then pass to the DBMS for processing.

By using the automatic translation abilities, you can often achieve the performance benefits of the SQL pass-through facility without needing to write DBMS-specific SQL code. The following sections describe the SAS SQL operations that SAS/ACCESS can pass to the DBMS for processing. For information about passing WHERE clauses to the DBMS, see “Optimizing the Passing of WHERE Clauses to the DBMS”.

Passing Functions to the DBMS Using PROC SQL
Overview of Passing Functions to a DBMS
Functions Where Results Might Vary: MOD Function
Passing Joins to the DBMS
Passing the DELETE Statement to Empty a Table
When Passing Joins to the DBMS Will Fail
Passing DISTINCT and UNION Processing to the DBMS
Optimizing the Passing of WHERE Clauses to the DBMS
General Guidelines for WHERE Clauses
Passing Functions to the DBMS Using WHERE Clauses
Using the DBINDEX=, DBKEY=, and MULTI_DATASRC_OPT= Options
Note: Certain conditions prevent operations from being passed to the DBMS. For example, when you use an INTO clause or any data set option, operations are processed in SAS instead of being passed to the DBMS. Re-merges, union joins, and truncated comparisons also prevent operations from being passed to the DBMS. Also, when you join tables across multiple tables, implicit pass-through uses the first connection. Consequently, LIBNAME options from subsequent connections are ignored. You can use the SASTRACE= system option to determine whether SAS processes an operation or whether pass-through passes it to the DBMS for processing.

To prevent operations from being passed to the DBMS, use the DIRECT_SQL= LIBNAME option.

---

Passing Functions to the DBMS Using PROC SQL

Overview of Passing Functions to a DBMS

When you use the SAS/ACCESS LIBNAME statement, it automatically tries to pass the SAS SQL aggregate functions (MIN, MAX, AVG, MEAN, FREQ, N, SUM, and COUNT) to the DBMS because these are SQL ANSI-defined aggregate functions.

Here is a sample query of the Oracle EMP table that is being passed to the DBMS for processing.

```sas
libname myoralib oracle user=myusr1 password=mypwd1;
proc sql;
  select count(*) from myoralib.emp;
quit;
```

This code causes Oracle to process this query.

```sql
select COUNT(*) from EMP
```

When a function is located in the WHERE clause of the SQL statement, SAS/ACCESS can translate other SAS functions into DBMS-specific functions so that pass-through can pass them to the DBMS.

In this example, the SAS UPCASE function is translated into the Oracle UPPER function.

```sas
libname myoralib oracle user=myusr1 password=mypwd1;
proc sql;
  select customer from myoralib.customers
  where upcase(country)="USA";
quit;
```

Here is the translated query that Oracle processes.

```sql
select customer from customers where upper(country)='USA'
```

Functions that pass-through are different for each DBMS. See the SQL pass-through section in the DBMS-specific reference for your SAS/ACCESS interface for a list of functions that it translates.

Functions Where Results Might Vary: MOD Function

In general, SAS functions that are passed through to a DBMS yield the same results. That is, your results are the same whether you pass a function to SAS or to your DBMS.
However, in some cases, SAS might yield a different result than you would obtain with the same function call to your DBMS.

The MOD function is one function that might give a different result depending on where the function runs. More specifically, when you pass non-integer arguments to the MOD function, the results might differ. The MOD function returns the remainder from the division of the first argument by the second argument. In SAS, both arguments can be non-integers, and the calculations are performed without altering the arguments. In some DBMSs, such as DB2 or PostgreSQL, non-integer arguments are truncated to the nearest integer before performing the division.

For example, the result of \( \text{MOD}(9, 2.25) \) is 0 in SAS, because 2.25 divides evenly into 9. The same call in PostgreSQL results in a value of 1, because 2.25 is first truncated to 2. The division of 9 by 2 then results in a value of 1 in PostgreSQL.

As a best practice, be certain that you understand how the MOD function works on your DBMS. Otherwise, you might get unexpected results.

---

**Passing Joins to the DBMS**

When you perform a join across SAS/ACCESS librefs in a single DBMS, PROC SQL can often pass the join to the DBMS for processing. Before implementing a join, PROC SQL checks to see whether the DBMS can process the join. A comparison is made using the SAS/ACCESS LIBNAME statement on page 96 for the librefs. Certain criteria must be met for the join to proceed. See the passing joins section in the DBMS-specific reference for your SAS/ACCESS interface to see the criteria that it requires before PROC SQL can pass the join.

If it can, PROC SQL passes the join to the DBMS. The DBMS then performs the join and returns only the results to SAS. PROC SQL processes the join if the DBMS cannot.

These types of joins are eligible for passing to the DBMS.

- For all DBMSs, inner joins between two or more tables.
- For DBMSs that support ANSI outer join syntax, outer joins between two or more DBMS tables.
- For ODBC and Microsoft SQL Server, outer joins between two or more tables. However, the outer joins must not be mixed with inner joins in a query.
- For such DBMSs as Informix, Oracle, and Sybase that support nonstandard outer join syntax, outer joins between two or more tables with these restrictions:
  - Full outer joins are not supported.
  - Only a comparison operator is allowed in an ON clause. For Sybase, the only valid comparison operator is ‘=’.
  - For Oracle and Sybase, both operands in an ON clause must reference a column name. A literal operand cannot be passed to the DBMS. Because these DBMSs do not support this, all ON clauses are transformed into WHERE clauses before trying to pass the join to the DBMS. This can result in queries not being passed to the DBMS if they include additional WHERE clauses or contain complex join conditions.
  - For Informix, outer joins can neither consist of more than two tables nor contain a WHERE clause.
Sybase evaluates multijoins with WHERE clauses differently than SAS. Therefore, instead of passing multiple joins or joins with additional WHERE clauses to the DBMS, use the SAS/ACCESS DIRECT_SQL= LIBNAME option to let PROC SQL process the join internally.

*Note:* If PROC SQL cannot successfully pass down a complete query to the DBMS, it might try again to pass down a subquery. You can analyze the SQL that is passed to the DBMS by turning on SAS tracing options. SAS trace information displays the exact queries that are being passed to the DBMS for processing.

In this example, TABLE1 and TABLE2 are large DBMS tables. Each has a column named DeptNo, and the value for with equal values. You want to retrieve the rows from an inner join of these tables. PROC SQL detects the join between two tables in the DLBLIB library (which references an Oracle database), and SAS/ACCESS passes the join directly to the DBMS. The DBMS processes the inner join between the two tables and returns only the resulting rows to SAS.

```sql
libname dlib oracle user=myusr1 password=mypwd1;
proc sql;
  select tab1.deptno, tab1.dname from
dlib.table1 tab1,
dlib.table2 tab2
  where tab1.deptno = tab2.deptno;
quit;
```

The query is passed to the DBMS and generates this Oracle code.

```sql
select table1."deptno", table1."dname" from TABLE1, TABLE2
  where TABLE1."deptno" = TABLE2."deptno"
```

In this example, an outer join between two Oracle tables, TABLE1 and TABLE2, is passed to the DBMS for processing.

```sql
libname myoralib oracle user=myusr1 password=mypwd1;
proc sql;
  select * from myoralib.table1 right join myoralib.table2
    on table1.x = table2.x
  where table2.x > 1;
quit;
```

The query is passed to the DBMS and generates this Oracle code.

```sql
select table1."X", table2."X" from TABLE1, TABLE2
  where TABLE1."X" (+) = TABLE2."X"
  and (TABLE2."X" > 1)
```

### Passing the DELETE Statement to Empty a Table

When you use the SAS/ACCESS LIBNAME statement with the DIRECT_EXE option set to DELETE, the SAS SQL DELETE statement is passed to the DBMS for execution as long as it contains no WHERE clause. The DBMS deletes all rows but does not delete the table itself.

This example shows how a DELETE statement is passed to Oracle to empty the EMP table.

```sql
libname myoralib oracle user=myusr1 password=mypwd1 direct_exe=delete;
```

```sql
select * from myoralib.EMP delete;
```

The query is passed to the DBMS and generates this Oracle code.

```sql
delete from TABLE1
```

This example illustrates the use of the DELETE statement with the DIRECT_EXE option set to DELETE.
When Passing Joins to the DBMS Will Fail

By default, SAS/ACCESS tries to pass certain types of SQL statements directly to the DBMS for processing. Most notable are SQL join statements that would otherwise be processed as individual queries to each data source that belonged to the join. In that instance, PROC SQL would then perform the join internally. Passing the join to the DBMS for direct processing can result in significant performance gains.

However, there are several reasons why a join statement under PROC SQL might not be passed to the DBMS for processing. In general, the success of the join depends on the nature of the SQL that was coded and the DBMS's acceptance of the generated syntax. It is also greatly influenced by the use of option settings. Here are the primary reasons why join statements might fail to be passed.

The DBMS does not accept the generated SQL syntax.
PROC SQL attempts to pass the SQL join query directly to the DBMS for processing. The DBMS can reject the syntax for any number of reasons. In this event, PROC SQL attempts to open both tables individually and perform the join internally.

The SQL query involves multiple librefs that do not share connection characteristics.
If the librefs are specified using different servers, user IDs, or any other connection options, PROC SQL does not attempt to pass the statement to the DBMS for direct processing.

Using data set options in the query
Specifying any data set option on a table that is referenced in the SQL query prohibits the statement from successfully passing to the DBMS for direct processing.

Using certain LIBNAME options
Specifying LIBNAME options that request such member-level controls as table locks (READ_LOCK_TYPE= or UPDATE_LOCK_TYPE=) prohibits the statement from successfully passing to the DBMS for direct processing.

Using SAS functions on the SELECT clause
Specifying SAS functions on the SELECT clause can prevent joins from being passed.

Using the DIRECT_SQL= LIBNAME option setting
The default setting for the DIRECT_SQL= LIBNAME option is YES. PROC SQL attempts to pass SQL joins directly to the DBMS for processing. Other settings for DIRECT_SQL= influence the nature of the SQL statements that PROC SQL tries to pass down to the DBMS or if it tries to pass anything at all.

DIRECT_SQL=YES
PROC SQL automatically attempts to pass the SQL join query to the DBMS. This is the default setting for this option. The join attempt could fail due to a DBMS return code. If this happens, PROC SQL attempts to open both tables individually and perform the join internally.
DIRECT_SQL=NO
PROC SQL does not attempt to pass SQL join queries to the DBMS. However, other SQL statements can be passed. If the `MULTI_DATASRC_OPT=` is in effect, the generated SQL can also be passed.

DIRECT_SQL=NONE
PROC SQL does not attempt to pass any SQL directly to the DBMS for processing.

DIRECT_SQL=NOWHERE
PROC SQL attempts to pass SQL to the DBMS including SQL joins. However, it does not pass any WHERE clauses associated with the SQL statement. This causes any join that is attempted with direct processing to fail.

DIRECT_SQL=NOFUNCTIONS
PROC SQL does not pass any statements in which any function is present to the DBMS. Normally PROC SQL attempts to pass down any functions coded in the SQL to the DBMS, provided the DBMS supports the given function.

DIRECT_SQL=NOGENSQL
PROC SQL does not attempt to pass SQL join queries to the DBMS. Other SQL statements can be passed down, however. If the `MULTI_DATASRC_OPT=` is in effect, the generated SQL can be passed.

DIRECT_SQL=NOMULTOUTJOINS
PROC SQL does not attempt to pass any multiple outer joins to the DBMS for direct processing. Other SQL statements can be passed, however, including portions of a multiple outer join.

---

**Passing DISTINCT and UNION Processing to the DBMS**

When you use the SAS/ACCESS LIBNAME statement to access DBMS data, the DISTINCT and UNION operators are processed in the DBMS rather than in SAS. For example, when PROC SQL detects a DISTINCT operator, it passes the operator to the DBMS to check for duplicate rows. The DBMS then returns only the unique rows to SAS.

In this example, the CUSTBASE Oracle table is queried for unique values in the STATE column.

```sas
libname myoralib oracle user=myusr1 password=mypwd1;
proc sql;
   select distinct state from myoralib.custbase;
quit;
```

The DISTINCT operator is passed to Oracle and generates this Oracle code.

```sql
select distinct custbase."STATE" from CUSTBASE
```

Oracle then passes the results from this query back to SAS.
Optimizing the Passing of WHERE Clauses to the DBMS

General Guidelines for WHERE Clauses

Follow the general guidelines in this table for writing efficient WHERE clauses.

Table 5.1 Efficient WHERE Clause Guidelines

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Inefficient</th>
<th>Efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avoid the NOT operator if you can use an equivalent form.</td>
<td>Inefficient: where zipcode not&gt;8000</td>
<td>Efficient: where zipcode&lt;=8000</td>
</tr>
<tr>
<td>Avoid LIKE predicates that begin with % or _ .</td>
<td>Inefficient: where COUNTRY like '%INA'</td>
<td>Efficient: where COUNTRY like 'A %INA'</td>
</tr>
<tr>
<td>Avoid arithmetic expressions in a predicate.</td>
<td>Inefficient: where SALARY&gt;12*4000.00</td>
<td>Efficient: where SALARY&gt;48000.00</td>
</tr>
</tbody>
</table>

Whenever possible, SAS/ACCESS passes WHERE clauses to the DBMS, because the DBMS processes them more efficiently than SAS does. SAS translates the WHERE clauses into generated SQL code. The performance impact can be particularly significant when you are accessing large DBMS tables. The following section describes how and when functions are passed to the DBMS. For information about passing processing to the DBMS when you are using PROC SQL, see “Overview: Optimizing Your SQL Usage” on page 43.

If you have NULL values in a DBMS column that is used in a WHERE clause, be aware that your results might differ depending on whether the WHERE clause is processed in SAS or is passed to the DBMS for processing. This is because DBMSs tend to remove NULL values from consideration in a WHERE clause, but SAS does not.

To prevent WHERE clauses from being passed to the DBMS, use the DIRECT_SQL=NOWHERE LIBNAME option.

Passing Functions to the DBMS Using WHERE Clauses

When you use the SAS/ACCESS LIBNAME statement, SAS/ACCESS translates several SAS functions in WHERE clauses into DBMS-specific functions so that they can be passed to the DBMS.

In this SAS code, SAS can translate the FLOOR function into a DBMS function and pass the WHERE clause to the DBMS.

```sas
libname myoralib oracle user=myusr1 password=mypwd1;
proc print data=myoralib.personnel;
```
where floor(hourlywage)+floor(tips)<10;
run;

Generated SQL that the DBMS processes would be similar to this code.

```
SELECT "HOURLYWAGE", "TIPS" FROM PERSONNEL
WHERE ((FLOOR("HOURLYWAGE") + FLOOR("TIPS")) < 10)
```

If the WHERE clause contains a function that SAS cannot translate into a DBMS function, SAS retrieves all rows from the DBMS and applies the WHERE clause.

The functions that are passed are different for each DBMS. See the DBMS-specific reference section for your SAS/ACCESS interface to determine which functions it translates.

---

**Using the DBINDEX=, DBKEY=, and MULTI_DATASRC_OPT= Options**

When you code a join operation in SAS and the join cannot be passed directly to a DBMS for processing, SAS performs the join. Normally, this processing involves individual queries to each data source that belonged to the join, and SAS performs the join internally. When you join a large DBMS table and a small SAS data set or DBMS table using the DBKEY=, DBINDEX=, and MULTI_DATASRC_OPT= options might enhance performance. These options let you retrieve a subset of the DBMS data into SAS for the join.

When you specify MULTI_DATASRC_OPT=IN_CLAUSE for DBMS data sources in a PROC SQL join operation, the procedure retrieves the unique values of the join column from the smaller table to construct an IN clause. SAS uses this IN clause when SAS retrieves data from the larger DBMS table. SAS performs the join. If you use a SAS data set, no matter how large it is, it is always in the IN_CLAUSE. For better performance, it is recommended that the SAS data set be smaller than the DBMS table. If not, processing can be extremely slow.

MULTI_DATASRC_OPT= generates a SELECT COUNT to determine the size of data sets that are not SAS data sets. If you know the size of your data set, you can use the DBMASTER= data set option to designate the larger table.

MULTI_DATASRC_OPT= might provide performance improvements over DBKEY=. If you specify options, DBKEY= overrides MULTI_DATASRC_OPT=.

MULTI_DATASRC_OPT= is used only when SAS is processing a join with PROC SQL. It is not used for SAS DATA step processing. For certain join operations, such as those involving additional subsetting applying to the query, PROC SQL might determine that it is more efficient to process the join internally. In these situations, it does not use the MULTI_DATASRC_OPT= optimization even when specified. If PROC SQL determines that it can pass the join directly to the DBMS, it also does not use this option even though it is specified.

In this example, the MULTI_DATASRC_OPT= option is used to improve the performance of an SQL join statement. MULTI_DATASRC_OPT= instructs PROC SQL to pass the WHERE clause to the SAS/ACCESS engine with an IN clause built from the SAS table. The engine then passes this optimized query to the DBMS server. The IN clause is built from the unique values of the SAS DeptNo variable. As a result, only rows that match the WHERE clause are retrieved from the DBMS. Without this option, PROC SQL retrieves all rows from the Dept table and applies the WHERE clause during...
PROC SQL processing in SAS. Processing can be both CPU-, input-, and output-intensive if the Dept Oracle table is large.

```sas
data keyvalues;
  deptno=30;
  output;
  deptno=10;
  output;
run;
```

```sas
libname dblib oracle user=myusr1 password=mypwd1 path='myorapath' multi_datasrc_opt=in_clause;
```

```sas
proc sql;
  select bigtab.deptno, bigtab.loc
  from dblib.dept bigtab,
  keyvalues smalllds
  where bigtab.deptno=smalllds.deptno;
quit;
```

The SQL statement that SAS/ACCESS creates and passes to the DBMS is similar to this one.

```sql
SELECT "DEPTNO", "LOC" FROM DEPT WHERE ("DEPTNO" IN (10,30))
```

Using DBKEY or DBINDEX decreases performance when the SAS data set is too large. These options cause each value in the transaction data set to generate a new result set (or open cursor) from the DBMS table. For example, if your SAS data set has 100 observations with unique key values, you request 100 result sets from the DBMS, which might be very expensive. Determine whether use of these options is appropriate, or whether you can achieve better performance by reading the entire DBMS table (or by creating a subset of the table).

DBINDEX= and DBKEY= are mutually exclusive. If you specify them together, DBKEY= overrides DBINDEX=. Both of these options are ignored if you specify the SAS/ACCESS DBCONDITION= data set option or the SAS WHERE= data set option.

DBKEY= does not require that any database indexes be defined; nor does it check the DBMS system tables. This option instructs SAS to use the specified DBMS column name or names in the WHERE clause that is passed to the DBMS in the join.

To improve the performance of joins, you can also use the DBKEY= option in a SAS DATA step, with the KEY= option in the SET statement. In this case, specify a value of KEY= DBKEY. The DATA step below creates a new data file by joining the data file KEYVALUES with the DBMS table MYTABLE. It uses the variable DEPTNO with the DBKEY= option to cause SAS/ACCESS to issue a WHERE clause.

```sas
data sasuser.new;
  set sasuser.keyvalues;
  set dblib.mytable(dbkey=deptno) key=dbkey;
run;
```

**Note:** When you use DBKEY= with the DATA step MODIFY statement, there is no implied ordering of the data that is returned from the database. If the master DBMS table contains records with duplicate key values, using DBKEY= can alter the outcome of the DATA step. Because SAS regenerates result sets (open cursors) during transaction processing, the changes that you make during processing have an impact on the results of subsequent queries. Therefore, before you use DBKEY= in this context, determine whether your master DBMS file has duplicate values for
keys. Remember that the REPLACE, OUTPUT, and REMOVE statements can cause duplicate values to appear in the master table.

The DBKEY= option does not require or check for the existence of indexes created on the DBMS table. Therefore, the DBMS system tables are not accessed when you use this option. The DBKEY= option is preferred over the DBINDEX= option for this reason. If you perform a join and use PROC SQL, you must make sure that the columns that you specify through the DBKEY= option match the columns that you specify in the SAS data set.

**CAUTION:**

Before you use the DBINDEX= option, take extreme care to evaluate some characteristics of the DBMS data. The number of rows in the table, the number of rows returned in the query, and the distribution of the index values in the table are among the factors to take into consideration. Some experimentation might be necessary to discover the optimum settings.

You can use the DBINDEX= option instead of the DBKEY= option if you know that the DBMS table has one or more indexes that use the column(s) on which the join is being performed. Use DBINDEX=index-name if you know the name of the index, or use DBINDEX=YES if you do not know the name of the index. Use this option as a data set option, and not a LIBNAME option, because index lookup can potentially be an expensive operation.

DBINDEX= requires that the join table must have a database index that is defined on the columns involved in the join. If there is no index, all processing of the join takes place in SAS, where all rows from each table are read into SAS and SAS performs the join.

**Note:** NULLCHAR= and NULLCHARVAL= data set options determine how SAS missing character values are handled during DBINDEX= and DBKEY= processing.
Overview: Threaded Reads in SAS/ACCESS

In SAS 8 and earlier, the software opened a single connection to the DBMS to read a table. SAS statements requesting data were converted to an SQL statement and passed to the DBMS. The DBMS processed the SQL statement, produced a result set consisting of table rows and columns, and transferred the result set back to SAS on the single connection.

With a threaded Read, you can reduce the table read time by retrieving the result set on multiple connections between SAS and the DBMS. SAS can create multiple threads, and a read connection is established between the DBMS and each SAS thread. The result set is partitioned across the connections, and rows are passed to SAS simultaneously (in parallel) across the connections, which improves performance.
Underlying Technology of Threaded Reads

To perform a threaded Read, SAS first creates threads within the SAS session. Threads are standard operating system tasks that SAS controls. SAS then establishes a DBMS connection on each thread, causes the DBMS to partition the result set, and reads one partition per thread. To cause the partitioning, SAS appends a WHERE clause to the SQL so that a single SQL statement becomes multiple SQL statements, one for each thread. Here is an example.

```sas
proc reg SIMPLE
  data=dblib.salesdata (keep=salesnumber maxsales);
  ar _ALL_;
  run;
```

Previous versions of SAS opened a single connection and issued:

```
SELECT salesnumber,maxsales FROM SALESDATA;
```

Assuming that SalesData has an EmployeeNum integer column, SAS 9.1 might open two connections by issuing these two statements:

```
SELECT salesnumber,maxsales FROM salesdata WHERE (EMPLOYEENUM mod 2)=0;
SELECT salesnumber,maxsales FROM SALESData WHERE (EMPLOYEENUM mod 2)=1;
```

For more information about MOD, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

*Note:* *Might* is an important word here. Most but not all SAS/ACCESS interfaces support threaded Reads in SAS 9.1. The partitioning WHERE clauses that SAS generates vary. In cases where SAS cannot always generate partitioning WHERE clauses, the SAS user can supply them. In addition to WHERE clauses, other ways to partition data might also exist.

SAS/ACCESS Interfaces and Threaded Reads

Threaded Reads work across all UNIX and Windows platforms where you run SAS. For more information about threaded Reads, see the DBMS-specific reference section for your specific SAS/ACCESS interface.

Scope of Threaded Reads

SAS steps called threaded applications are automatically eligible for a threaded Read. Threaded applications are bottom-to-top fully threaded SAS procedures that perform data reads, numerical algorithms, and data analysis in threads. Only some SAS procedures are threaded applications. Here is a basic example of PROC REG, a SAS threaded application.

```sas
libname lib oracle user=myusr1 password=mypwd1;
proc reg simple
```
data=lib.salesdata (keep=salesnumber maxsales);
var _all_
run;

For DBMSs, many more SAS steps can become eligible for a threaded Read, specifically, steps with a read-only table. A libref has the form Lib.DbTable, where Lib is a SAS libref that "points" to DBMS data, and DbTable is a DBMS table. Here are sample read-only tables for which threaded Reads can be turned on.

libname lib oracle user=myusr1 password=mypwd1;
proc print data=lib.dbtable;
run;

data local;
set lib.families;
where gender="F"
run;

An eligible SAS step can require user assistance to actually perform threaded Reads. If SAS cannot automatically generate a partitioning WHERE clause or otherwise perform threaded Reads, the user can code an option that supplies partitioning. To determine whether SAS can automatically generate a partitioning WHERE clause, use the SASTRACE= and SASTRACELOC= system options.

Threaded Reads can be turned off altogether. This eliminates additional DBMS activity associated with SAS threaded Reads, such as additional DBMS connections and multiple SQL statements.

Threaded Reads are not supported for the pass-through facility, in which you code your own DBMS-specific SQL that is passed directly to the DBMS for processing.

Options That Affect Threaded Reads

For threaded Reads from DBMSs, SAS/ACCESS provides the DBSLICE= and DBSLICEPARM= data set options.

DBSLICE= applies only to a table reference. You can use it to code your own WHERE clauses to partition table data across threads, and it is useful when you are familiar with your table data. For example, if your DBMS table has a CHAR(1) column Gender and your clients are approximately half female, Gender equally partitions the table into two parts. Here is an example.

proc print data=lib.dbtable (dbslice=("gender='f'", "gender='m'"));
where dbcol>1000;
run;

SAS creates two threads and about half of the data is delivered in parallel on each connection.

When applying DBSLICEPARM=ALL instead of DBSLICE=, SAS attempts to "autopartition" the table for you. With the default DBSLICEPARM=THREADED_APPS setting, SAS automatically attempts threaded Reads only for SAS threaded applications. SAS threaded applications are SAS procedures that thread input, output, and numeric operations. DBSLICEPARM=ALL extends threaded Reads to more SAS procedures, specifically steps that only read tables. Or, DBSLICEPARM=NONE turns it off entirely. You can specify it as a data set option, a LIBNAME option, or a global SAS option.
The first argument to DBSLICEPARM= is required and extends or restricts threaded Reads. The second optional argument is not commonly used and limits the number of DBMS connections. These examples demonstrate the different uses of DBSLICEPARM=.

- UNIX or Windows SAS invocation option that turns on threaded Reads for all read-only libref:
  
  -dbsliceparm ALL

- Global SAS option that turns off threaded Reads:

  option dbsliceparm=NONE;

- LIBNAME option that restricts threaded Reads to just SAS threaded applications:

  libname lib oracle user=myusr1 password=mypwd1 dbsliceparm=THREADED_APPS;

- Table option that turns on threaded Reads, with a maximum of three connections in this example:

  proc print data=lib.dbtable(dbsliceparm=(ALL,3));
  where dbcol>1000;
  run;

DBSLICE= and DBSLICEPARM= apply only to DBMS table reads. THREADS= and CPUCOUNT= are additional SAS system options that apply to threaded applications. For more information about these options, see SAS System Options: Reference.

---

**Generating Trace Information for Threaded Reads**

A threaded Read is a complex feature. A SAS step can be eligible for a threaded Read, but not have it applied. Performance effect is not always easy to predict. Use the SASTRACE option to see whether threaded Reads occurred and to help assess performance. These examples demonstrate usage scenarios with SAS/ACCESS to Oracle. Keep in mind that trace output is in English only and changes from release to release.

/*Turn on SAS tracing */
options sastrace=",,t," sastraceloc=saslog nostsuffix;

/* Run a SAS job */

data work.locemp;
set trlib.MYEMPS(DBSLICEPARM=(ALL,3));
where STATE in ('GA', 'SC', 'NC') and ISTENURE=0;
run;

The above job produces these trace messages:

406  data work.locemp;
407  set trlib.MYEMPS(DBSLICEPARM=(ALL, 3));
408  where STATE in ('GA', 'SC', 'NC') and ISTENURE=0;
409  run;

ORACLE:  DBSLICEPARM option set and 3 threads were requested
ORACLE:  No application input on number of threads.

ORACLE:  Thread 2 contains 47619 obs.
If you want to see the SQL that is executed during the threaded Read, you can set tracing to sastrace=',t,d' and run the job again. This time the output contains threading information and all SQL statements that Oracle processes.

```
ORACLE_9: Prepared:
SELECT * FROM MYEMPS  418  data work.locemp;
419  set trlib.MYEMPS(DBSLICEPARM=(ALL, 3));
420  where STATE in ('GA', 'SC', 'NC') and ISTENURE=0;
421  run;

ORACLE: DBSLICEPARM option set and 3 threads were requested
```

Notice that the Oracle engine used the EMPNUM column as a partitioning column. If a threaded Read cannot be done either because all of the candidates for autopartitioning are in the WHERE clause, or because the table does not contain a column that fits the criteria, SAS places a warning in the log. For example, the data set below uses a WHERE clause that contains all possible autopartitioning columns.

```
data work.locemp;
set trlib.MYEMPS (DBSLICEPARM=ALL);
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASSES>2;
run;
```

These warnings are displayed:

```
ORACLE: WARNING: Unable to find a partition column for use w/ MOD()
```

The engine cannot automatically generate the partitioning WHERE clauses.
ORACLE: Using only one read connection.
ORACLE: Threading is disabled due to an error. Application reverts to nonthreading I/O's.

If the SAS job contains any options that are invalid when the engine tries to perform threading, you also receive a warning.

```
libname trlib oracle user=myusr1 pw=mypwd1 path=oraserver DBSLICEPARM=(ALL);

proc print data=trlib.MYEMPS (OBS=10);
where EMPNUM<=30;
run;
```

This produces these messages:

```
ORACLE: Threading is disabled due to the ORDER BY clause or the FIRSTOBS/OBS option.
ORACLE: Using only one read connection.
```

To produce timing information, add an S in the last slot of SASTRACE=.

```
options sastrace=',,t,s' sastraceloc=saslog nostsuffix;
```

```
data work.locemp;
set trlib.MYEMPS (DBSLICEPARM=ALL);
where EMPNUM<=10000;
run;
```

Here is the resulting timing information.

```
ORACLE: No application input on number of threads.
ORACLE: Thread 1 contains 5000 obs.
ORACLE: Thread 2 contains 5000 obs.

Thread 0 fetched 5000 rows
DBMS Threaded Read Total Time: 1234 mS
DBMS Threaded Read User CPU:  46 mS
DBMS Threaded Read System CPU:  0 mS

Thread 1 fetched 5000 rows
DBMS Threaded Read Total Time: 469 mS
DBMS Threaded Read User CPU:  15 mS
DBMS Threaded Read System CPU:  15 mS
ORACLE: Threaded read enabled. Number of threads created: 2
NOTE: There were 10000 observations read from the data set TRLIB.MYEMPS.
WHERE EMPNUM<=10000;
```

Summary Statistics for ORACLE are: Total SQL prepare seconds were:  0.00167
Total seconds used by the ORACLE ACCESS engine were 7.545805

For more information, see the SASTRACE= system option on page 455.

---

**Performance Impact of Threaded Reads**

Threaded Reads only increase performance when the DBMS result set is large. Performance is optimal when the partitions are similar in size. Using threaded Reads should reduce the elapsed time of your SAS step, but unusual cases can slow the SAS step. They generally increase the workload on your DBMS.
For example, threaded Reads for DB2 under z/OS involve a tradeoff, generally reducing job elapsed time but increasing DB2 workload and CPU usage. See the auto partitioning documentation for DB2 under z/OS for details.

SAS automatically tries to autopartition table references for SAS in threaded applications. To determine whether autopartitioning is occurring and to assess its performance, complete these tasks.

- Turn on SAS tracing to see whether SAS is autopartitioning and to view the SQL associated with each thread.
- Know your DBMS algorithm for autopartitioning.
- Turn threaded Reads on and off, and compare the elapsed times.

Follow these guidelines to ensure optimal tuning of threaded Reads.

- Use it only when pulling large result sets into SAS from the DBMS.
- Use DBSLICE= to partition if SAS autopartitioning does not occur.
- Override autopartitioning with DBSLICE= if you can manually provide substantially better partitioning. The best partitioning equally distributes the result set across the threads.
- See the DBMS-specific reference section in this document for information and tips for your DBMS.

Threaded Reads are most effective on new, faster computer hardware running SAS, and with a powerful parallel edition of the DBMS. For example, if SAS runs on a fast uniprocessor or on a multiprocessor machine and your DBMS runs on a high-end SMP server, you can experience substantial performance gains. However, you can experience minimal gains or even performance degradation when running SAS on an old desktop model with a nonparallel DBMS edition running on old hardware.

Autopartitioning Techniques in SAS/ACCESS

SAS/ACCESS products share an autopartitioning scheme based on the MOD function. Some products support additional techniques. For example, if your Oracle tables are physically partitioned in the DBMS, SAS/ACCESS Interface to Oracle automatically partitions in accordance with Oracle physical partitions rather than using MOD. SAS/ACCESS Interface to Teradata uses FastExport, if available, which lets the FastExport Utility direct partitioning.

MOD is a mathematical function that produces the remainder of a division operation. Your DBMS table must contain a column to which SAS can apply the MOD function—a numeric column constrained to integral values. DBMS integer and small integer columns suit this purpose. Integral decimal (numeric) type columns can work as well. On each thread, SAS appends a WHERE clause to your SQL that uses the MOD function with the numeric column to create a subset of the result set. Combined, these subsets add up to exactly the result set for your original single SQL statement.

For example, assume that your original SQL that SAS produces is

```
SELECT CHR1, CHR2 FROM DBTAB
```

and that the Dbtab table contains the IntCol integer column. SAS creates two threads and issues these two statements:

```
SELECT CHR1, CHR2 FROM DBTAB WHERE (MOD(INTCOL,2)=0)
```

```
SELECT CHR1, CHR2 FROM DBTAB WHERE (MOD(INTCOL,2)=1)
```
The first thread retrieves rows with an even value for IntCol, and the second thread retrieves rows with an odd value for IntCol. Distribution of rows across the two threads is optimal if IntCol has a 50/50 distribution of even and odd values.

SAS modifies the SQL for columns containing negative integers, for nullable columns, and to combine SAS WHERE clauses with the partitioning WHERE clauses. SAS can also run more than two threads. You use the second parameter of the DBSLICEPARM= option to increase the number of threads.

The success of this technique depends on the distribution of the values in the chosen integral column. Without knowledge of the distribution, your SAS/ACCESS product attempts to select the best possible column. For example, indexed columns are given preference for some DBMSs. However, column selection is more or less a guess, and the SAS guess might cause poor distribution of the result set across the threads. If no suitable numeric column is found, MOD cannot be used at all, and threaded Reads do not occur if your SAS/ACCESS product has no other partitioning technique. For these reasons, you should explore autopartitioning particulars for your DBMS and judiciously use DBSLICE= to augment autopartitioning. See the information for your DBMS for specific autopartitioning details.

- Aster on page 494
- DB2 under UNIX and PC Hosts on page 513
- DB2 under z/OS on page 540
- Greenplum on page 592
- Informix on page 671
- ODBC on page 743
- Oracle on page 789 (not supported under z/OS)
- SAP HANA on page 838
- Sybase on page 856
- Sybase IQ on page 881
- Teradata: For platform-specific details and special considerations, see your Teradata documentation and also “Autopartitioning Scheme for Teradata (Legacy)” on page 914.

---

Data Ordering in SAS/ACCESS

The order in which table rows are delivered to SAS varies each time a step is rerun with threaded Reads. Most DBMS editions, especially increasingly popular parallel editions, do not guarantee consistent ordering.

---

Two-Pass Processing for SAS Threaded Applications

Two-pass processing occurs when a request is received for data to be made available for multiple pass reading (that is, more than one pass through the data set). In the context of DBMS engines, this requires that as the data is read from the database, temporary spool
files are written containing the read data. There is one temporary spool file per thread, and each spool file contains all data read on that thread. If three threads are specified for threaded Reads, three temporary spool files are written.

As the application requests subsequent passes of data, data is read from the temporary spool files, not reread from the database. The temporary spool files can be written on different disks, reducing any disk read contention, and enhancing performance. To accomplish this, the SAS option UTILLOC= is used to define different disk devices and directory paths when creating temporary spool files. There are several ways to specify this option.

- In the SAS config file, add this line:
  
  \texttt{-utilloc("C:\path" "D:\path" "E:\path")}

- Specify the UTILLOC= option on the SAS command line:
  
  /* on Windows */
  \texttt{sas -utilloc(c:\path d:\path e:\path)}

  /* on UNIX */
  \texttt{sas -utilloc ('\path \path2 \path3')}

For more information about the UTILLOC= SAS system option, see the \textit{SAS System Options: Reference}.

### When Threaded Reads Do Not Occur

Threading does not occur under these circumstances:

- when a BY statement is used in a PROC or DATA step
- when the OBS option or the FIRSTOBS option is in a PROC or DATA step
- when the KEY option or the DBKEY option is used PROC or DATA step
- if no column in the table exists to which SAS can apply the MOD function (for more information, see “Autopartitioning Techniques in SAS/ACCESS” on page 59)
- if all columns within a table to which SAS can apply the MOD function are in WHERE clauses (for more information, see “Autopartitioning Techniques in SAS/ACCESS” on page 59)
- if the NOTHREADS system option is set
- if DBSLICE=PARM=NONE

### Summary of Threaded Reads

For large reads of table data, SAS threaded Reads can speed up SAS jobs. They are particularly useful when you understand the autopartitioning technique specific to your DBMS and use DBSLICE= to manually partition only when appropriate. Look for enhancements in future SAS releases.
Chapter 7
National Language Support

Overview: NLS for SAS/ACCESS

National Language Support (NLS) is a set of features that enable a software product to function properly in every global market for which the product is targeted. SAS/ACCESS interfaces provide NLS through LIBNAME and SQL pass-through options and also through data types.

For NLS limitations that are specific to Hive, see “Naming Conventions for SAS and Hive”.

For more NLS information, see these titles.

- *SAS Encoding: Understanding the Details*, available from SAS Press books
- “Processing Multilingual Data with the SAS 9.2 Unicode Server”, a SAS Institute white paper

Options

These LIBNAME and SQL pass-through (connection) options allow for byte and character conversion and length calculation.

- `ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=`
- `ADJUST_NCHAR_COLUMN_LENGTHS=`
- `DB_LENGTH_SEMANTICS_BYTE=`
- `DBCLIENT_MAX_BYTES=`
- `DBSERVER_MAX_BYTES=`
The TRANSCODE_FAIL= LIBNAME and data set options for SAS/ACCESS Interface to Hadoop let you specify how to handle processing and notification of transcoding errors.

## Data Types

These data types allow for more flexible adjustment of column lengths.

- BLOB
- CHAR
- CLOB
- DBCLOB
- NCHAR
- VARCHAR
Chapter 8
How SAS/ACCESS Works

Introduction to How SAS/ACCESS Works

Installation Requirements

Before you use any SAS/ACCESS features, you must install Base SAS, the SAS/ACCESS interface for the DBMS that you are accessing, and any required DBMS client software. See SAS installation instructions and DBMS client installation instructions for more information.

Not all SAS/ACCESS interfaces support all features. See the DBMS-specific reference section for your SAS/ACCESS interface to determine which features are supported in your environment.

SAS/ACCESS Interfaces

Each SAS/ACCESS interface consists of one or more data access engines that translate read and write requests from SAS into appropriate calls for a specific DBMS. The following image depicts the relationship between a SAS/ACCESS interface and a relational DBMS.
You can call a SAS/ACCESS relational DBMS interface by using either a LIBNAME statement or a PROC SQL statement. (Although you can also use ACCESS and DBLOAD procedures with some of the SAS/ACCESS relational interfaces, these procedures are no longer the recommended way to access relational database data.)

How the SAS/ACCESS LIBNAME Statement Works

**Accessing Data from a DBMS Object**

You can use SAS/ACCESS to read, update, insert, and delete data from a DBMS object as if it were a SAS data set. Here are the steps.

1. Start a SAS/ACCESS interface by specifying a DBMS engine name and the appropriate connection options in a LIBNAME statement.
2. Enter SAS requests as you would when accessing a SAS data set.
3. SAS/ACCESS generates DBMS-specific SQL statements that are equivalent to the SAS requests that you enter.
4. SAS/ACCESS submits the generated SQL to the DBMS.

The SAS/ACCESS engine defines which operations are supported on a table and calls code that translates database operations such as open, get, put, or delete into DBMS-specific SQL syntax. SAS/ACCESS engines use an established set of routines with calls that are customized for each DBMS.
Processing Queries, Joins, and Data Functions

To enhance performance, SAS/ACCESS can transparently pass queries, joins, and data functions to the DBMS for processing instead of retrieving the data from the DBMS and processing it in SAS. For example, an important use of this feature is the handling of PROC SQL queries that access DBMS data. Here is how it works.

1. PROC SQL examines each query to determine whether it might be profitable to send all or part of the query to the DBMS for processing.
2. A special query textualizer in PROC SQL translates queries (or query fragments) into DBMS-specific SQL syntax.
3. The query textualizer submits the translated query to the SAS/ACCESS engine for approval.
4. If SAS/ACCESS approves the translation, it sends an approval message to PROC SQL. The DBMS processes the query or query fragment and returns the results to SAS. Any queries or query fragments that cannot be passed to the DBMS are processed in SAS.

For details about tasks that SAS/ACCESS can pass to the DBMS, see the DBMS-specific reference section for your SAS/ACCESS interface.

How the SQL Pass-Through Facility Works

When you read and update DBMS data with the SQL pass-through facility, SAS/ACCESS passes SQL statements directly to the DBMS for processing. Here are the steps.

1. Invoke PROC SQL and submit a PROC SQL CONNECT statement that includes a DBMS name and the appropriate connection options to establish a connection with a specified database.

2. Use a CONNECTION TO component in a PROC SQL SELECT statement to read data from a DBMS table or view.

   In the SELECT statement (PROC SQL query) that you write, use the SQL that is native to your DBMS. SAS/ACCESS passes the SQL statements directly to the DBMS for processing. If the SQL syntax that you enter is correct, the DBMS processes the statement and returns any results to SAS. If the DBMS does not recognize the syntax that you enter, it returns an error that appears in the SAS log. The SELECT statement can be stored as a PROC SQL view. Here is an example.

   ```
   proc sql;
   connect to oracle (user=myusr1 password=mypwd1);
   create view budget2000 as select amount_b,amount_s
   from connection to oracle
   (select Budgeted, Spent from annual_budget);
   quit;
   ```

3. Use a PROC SQL EXECUTE statement to pass any dynamic, nonquery SQL statements (such as INSERT, DELETE, and UPDATE) to the database.

   As with the CONNECTION TO component, all EXECUTE statements are passed to the DBMS exactly as you submit them. INSERT statements must contain literal values. Here is an example.
proc sql;
   connect to oracle(user=myusr1 password=mypwd1);
   execute (create view whotookorders as select ordernum, takenby,
          firstname, lastname, phone from orders, employees
          where orders.takenby=employees.empid) by oracle;
   execute (grant select on whotookorders to myusr1) by oracle;
   disconnect from oracle;
   quit;

4. End the connection with the DISCONNECT statement.

For more details, see the SQL pass-through facility on page 472.

How the ACCESS Procedure Works

Overview: ACCESS Procedure

When you use the ACCESS procedure to create an access descriptor, the SAS/ACCESS interface view engine requests the DBMS to execute an SQL SELECT statement to the data dictionary tables in your DBMS dynamically by using DBMS-specific call routines or interface software. The ACCESS procedure then issues the equivalent of a DESCRIBE statement to gather information about the columns in the specified table. Access descriptor information about the table and its columns is then copied into the view descriptor when it is created. It is therefore not necessary for SAS to call the DBMS when it creates a view descriptor. Here is the process.

1. When you provide connection information to PROC ACCESS, the SAS/ACCESS interface calls the DBMS to connect to the database.

2. SAS constructs a SELECT * FROM table-name statement and passes it to the DBMS to retrieve information about the table from the DBMS data dictionary. This SELECT statement is based on the information that you provided to PROC ACCESS. It lets SAS determine whether the table exists and can be accessed.

3. The SAS/ACCESS interface calls the DBMS to obtain table description information, such as the column names, data types (including width, precision, and scale), and whether the columns accept null values.

4. SAS closes the connection with the DBMS.

Reading Data

When you use a view descriptor in a DATA step or procedure to read DBMS data, the SAS/ACCESS interface view engine requests the DBMS to execute an SQL SELECT statement. Here are the steps that the interface view engine follows.

1. Using the connection information that is in the created view descriptor, the SAS/ACCESS interface calls the DBMS to connect to the database.

2. SAS constructs a SELECT statement that is based on the information stored in the view descriptor (table name and selected columns and their characteristics) and passes this information to the DBMS.
3. SAS retrieves the data from the DBMS table and passes it back to the SAS procedures as if it were observations in a SAS data set.

4. SAS closes the connection with the DBMS.

For example, if you run the following SAS program using a view descriptor, the previous steps are executed once for the PRINT procedure and a second time for the GCHART procedure. (The data that is used for the two procedures is not necessarily the same because another user might have updated the table might have been updated between procedure executions.)

```sas
proc print data=vlib.allemp;
run;

proc gchart data=vlib.allemp;
  vbar jobcode;
run;
```

---

### Updating Data

Use a view descriptor, DATA step, or procedure to update DBMS data similarly to when you read in data. Any of these steps might also occur.

1. Using the connection information that is contained in the specified access descriptor, the SAS/ACCESS interface calls the DBMS to connect to the database.

2. When rows are added to a table, SAS constructs an SQL INSERT statement and passes it to the DBMS. When you reference a view descriptor, use the ADD command in FSEDIT and FSVIEW, the APPEND procedure, or an INSERT statement in PROC SQL to add data to a DBMS table. (You can also use the EXECUTE statement for the SQL pass-through facility to add, delete, or modify DBMS data directly. You must use literal values when you insert data with the SQL pass-through facility.)

3. When rows are deleted from a DBMS table, SAS constructs an SQL DELETE statement and passes it to the DBMS. When you reference a view descriptor, you can use the DELETE command in FSEDIT and FSVIEW or a DELETE statement in PROC SQL to delete rows from a DBMS table.

4. When data in the rows is modified, SAS constructs an SQL UPDATE statement and passes it to the DBMS. When you reference a view descriptor, you can use FSEDIT, the MODIFY command in FSVIEW, or an INSERT statement in PROC SQL to update data in a DBMS table. You can also reference a view descriptor in the UPDATE, MODIFY, and REPLACE statements for the DATA step.

5. SAS closes the connection with the DBMS.

---

### How the DBLOAD Procedure Works

When you use the DBLOAD procedure to create a DBMS table, the procedure issues dynamic SQL statements to create the table and insert data from a SAS data file, DATA step view, PROC SQL view, or view descriptor into the table. Here are the steps that the SAS/ACCESS interface view engine completes.
1. When you supply the connection information to PROC DBLOAD, the SAS/ACCESS interface calls the DBMS to connect to the database.

2. SAS uses the information that the DBLOAD procedure provides to construct a SELECT * FROM table-name statement and passes the information to the DBMS to determine whether the table already exists. PROC DBLOAD continues only if a table with that name does not exist unless you use the DBLOAD APPEND option.

3. SAS uses the information that the DBLOAD procedure provides to construct an SQL CREATE TABLE statement and passes it to the DBMS.

4. SAS constructs an SQL INSERT statement for the current observation and passes it to the DBMS. New INSERT statements are constructed and executed repeatedly until all observations from the input SAS data set are passed to the DBMS. Some DBMSs have a bulk-copy capability so that a group of observations can be inserted at once. See your DBMS documentation to determine whether your DBMS has this capability.

5. Additional nonquery SQL statements that are specified in the DBLOAD procedure are executed as the user submitted them. The DBMS returns an error message if a statement does not execute successfully.

6. SAS closes the connection with the DBMS.
Chapter 9
In-Database Processing with SAS/ACCESS

Overview: In-Database Processing

When you use conventional processing to access data that is inside a DBMS, SAS asks the SAS/ACCESS engine for all table rows of the table that is being processed. The SAS/ACCESS engine generates an SQL SELECT * statement, which is passed to the DBMS. That SELECT statement fetches all rows in the table, and the SAS/ACCESS engine returns them to SAS. The number of rows in the table grows over time, so network latency grows because the amount of data that is fetched from the DBMS to SAS increases.

SAS In-Database processing integrates SAS solutions, SAS analytic processes, and third-party DBMSs. Using SAS In-Database processing, you can run scoring models, Base SAS and SAS/STAT procedures, and formatted SQL queries inside the database.

For more information, see SAS In-Database Products: Administrator’s Guide and SAS In-Database Products: User’s Guide.

Table 9.1 SAS Products Needed to Use In-Database Features

<table>
<thead>
<tr>
<th>In-Database Feature</th>
<th>Required Software</th>
<th>Supported DBMSs</th>
</tr>
</thead>
</table>
| format publishing and the SAS_PUT() function | • Base SAS  
• SAS/ACCESS DBMS interfaces | Aster  
DB2 (only on UNIX hosts)  
Greenplum  
Netezza  
Teradata |
<table>
<thead>
<tr>
<th>In-Database Feature</th>
<th>Required Software</th>
<th>Supported DBMSs</th>
</tr>
</thead>
<tbody>
<tr>
<td>scoring models</td>
<td>• Base SAS</td>
<td>Aster</td>
</tr>
<tr>
<td></td>
<td>• SAS/ACCESS DBMS interfaces</td>
<td>DB2 (only on UNIX hosts)</td>
</tr>
<tr>
<td></td>
<td>• SAS Scoring Accelerator</td>
<td>Greenplum</td>
</tr>
<tr>
<td></td>
<td>• SAS Enterprise Miner</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>• SAS Scalable Performance Data Server (optional)</td>
<td>Netezza</td>
</tr>
<tr>
<td></td>
<td>• SAS Model Manager (optional)</td>
<td>Oracle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP HANA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SPD Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Teradata</td>
</tr>
<tr>
<td>Base SAS procedures:</td>
<td>• Base SAS</td>
<td>Aster</td>
</tr>
<tr>
<td></td>
<td>• SAS/ACCESS DBMS interfaces</td>
<td>DB2</td>
</tr>
<tr>
<td></td>
<td>• SAS In-Database Code Accelerator [PROC TRANSPOSE only]</td>
<td>Greenplum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HAWQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Impala</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Netezza</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP HANA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Teradata</td>
</tr>
<tr>
<td>SAS/STAT procedures:</td>
<td>• Base SAS (for CORR)</td>
<td>Teradata</td>
</tr>
<tr>
<td></td>
<td>• SAS/ACCESS Interface to Teradata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAS/STAT (for CANCORR, FACTOR, PRINCOMP, REG, SCORE, VARCLUS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAS/ETS (for TIMESERIES)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAS Enterprise Miner (for DMDB, DMINE, DMREG)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAS Analytics Accelerator</td>
<td></td>
</tr>
<tr>
<td>DS2 threaded programs</td>
<td>• Base SAS</td>
<td>Greenplum</td>
</tr>
<tr>
<td></td>
<td>• SAS/ACCESS DBMS interfaces</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>• SAS In-Database Code Accelerator</td>
<td>Teradata</td>
</tr>
<tr>
<td>DATA step scoring programs</td>
<td>• Base SAS</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>• SAS/ACCESS Interface to Hadoop</td>
<td></td>
</tr>
<tr>
<td>In-Database Feature</td>
<td>Required Software</td>
<td>Supported DBMSs</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Data quality operations</td>
<td>• Base SAS</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>• SAS/ACCESS DBMS interfaces</td>
<td>Teradata</td>
</tr>
<tr>
<td></td>
<td>• SAS Data Loader for Hadoop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAS Data Quality Accelerator for Teradata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAS In-Database Code Accelerator for Teradata</td>
<td></td>
</tr>
<tr>
<td>Extract and transform data operations</td>
<td>• Base SAS</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>• SAS/ACCESS DBMS interfaces</td>
<td>Teradata</td>
</tr>
<tr>
<td></td>
<td>• SAS Data Loader for Hadoop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAS Data Quality Accelerator for Teradata</td>
<td></td>
</tr>
</tbody>
</table>
Part 2

General Reference

Chapter 10
SAS/ACCESS Features by Host ................................. 77

Chapter 11
The LIBNAME Statement for Relational Databases ............ 93

Chapter 12
Data Set Options for Relational Databases .................... 237

Chapter 13
Macro Variables and System Options for Relational Databases . 445

Chapter 14
SQL Pass-Through Facility for Relational Databases .......... 471
# SAS/ACCESS Features by Host

This section provides a quick summary, by host environment, of the features that are available for your SAS/ACCESS interface.

## Introduction

---

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>77</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Aster: Supported Features</td>
<td>78</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to DB2 under UNIX and PC Hosts: Supported Features</td>
<td>78</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to DB2 under z/OS: Supported Features</td>
<td>79</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Greenplum: Supported Features</td>
<td>79</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Hadoop: Supported Features</td>
<td>80</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to HAWQ: Supported Features</td>
<td>81</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Impala: Supported Features</td>
<td>82</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Informix: Supported Features</td>
<td>82</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Microsoft SQL Server: Supported Features</td>
<td>83</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to MySQL: Supported Features</td>
<td>83</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Netezza: Supported Features</td>
<td>84</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to ODBC: Supported Features</td>
<td>85</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to OLE DB: Supported Features</td>
<td>86</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Oracle: Supported Features</td>
<td>86</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to PostgreSQL: Supported Features</td>
<td>87</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to SAP HANA: Supported Features</td>
<td>88</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Sybase: Supported Features</td>
<td>88</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Sybase IQ: Supported Features</td>
<td>89</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Teradata: Supported Features</td>
<td>90</td>
</tr>
<tr>
<td>SAS/ACCESS Interface to Vertica: Supported Features</td>
<td>90</td>
</tr>
</tbody>
</table>
For detailed information for your particular interface, see the SAS system requirements and configuration guide documents, which are available at SAS Support.

SAS/ACCESS Interface to Aster: Supported Features

Here are the features that SAS/ACCESS Interface to Aster supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.1 Features by Host Environment for Aster

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Note: Support for the AIX and Solaris for SPARC platforms was added in the first maintenance release for SAS 9.4.

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading and Unloading for Aster” on page 497.

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts: Supported Features

Here are the features that SAS/ACCESS Interface to DB2 under UNIX and PC Hosts supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.2 Features by Host Environment for DB2 under UNIX and PC Hosts

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for DB2 under UNIX and PC Hosts” on page 521.

SAS/ACCESS Interface to DB2 under z/OS: Supported Features

Here are the features that SAS/ACCESS Interface to DB2 under z/OS supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.3  Features by Host Environment for DB2 under z/OS

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for DB2 under z/OS” on page 561.

SAS/ACCESS Interface to Greenplum: Supported Features

Here are the features that SAS/ACCESS Interface to Greenplum supports. To find out which versions of your DBMS are supported, see your system requirements documentation.
Table 10.4  Features by Host Environment for Greenplum

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for Greenplum” on page 595.

SAS/ACCESS Interface to Hadoop: Supported Features

Here are the features that SAS/ACCESS Interface to Hadoop supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.5  Features by Host Environment for Hadoop

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support*</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

* SAS/ACCESS to Hadoop does not differentiate between bulk loading and a standard load process.
For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for Hadoop” on page 613.

For detailed information about the following topics, see the SAS system requirements and configuration guide documents. They are available at http://support.sas.com:

- supported Hadoop versions, character data formats support, and required JAR files, environmental variables, and patches—along with other prerequisites, required setup, and considerations for SAS/ACCESS Interface to Hadoop
- how SAS/ACCESS Interface to Hadoop interacts with Hadoop through Hive

### SAS/ACCESS Interface to HAWQ: Supported Features

Here are the features that SAS/ACCESS Interface to HAWQ supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

#### Table 10.6  Features by Host Environment for HAWQ

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for HAWQ” on page 641.
SAS/ACCESS Interface to Impala: Supported Features

Here are the features that SAS/ACCESS Interface to Impala supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

**Table 10.7 Features by Host Environment for Impala**

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

SAS/ACCESS Interface to Informix: Supported Features

Here are the features that SAS/ACCESS Interface to Informix supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

**Table 10.8 Features by Host Environment for Informix**

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4.
SAS/ACCESS Interface to Microsoft SQL Server: Supported Features

Here are the features that SAS/ACCESS Interface to Microsoft SQL Server supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.9  Features by Host Environment for Microsoft SQL Server

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4.

SAS/ACCESS Interface to MySQL: Supported Features

Here are the features that SAS/ACCESS Interface to MySQL supports. To find out which versions of your DBMS are supported, see your system requirements documentation.
Table 10.10  Features by Host Environment for MySQL

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4.

SAS/ACCESS Interface to Netezza: Supported Features

Here are the features that SAS/ACCESS Interface to Netezza supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.11  Features by Host Environment for Netezza

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading and Unloading for Netezza” on page 721.

### SAS/ACCESS Interface to ODBC: Supported Features

Here are the features that SAS/ACCESS Interface to ODBC supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

**Table 10.12  Features by Host Environment for ODBC**

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

* Bulk-load support is available only with the Microsoft SQL Server driver on Microsoft Windows platforms.

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for ODBC” on page 750.
SAS/ACCESS Interface to OLE DB: Supported Features

Here are the features that SAS/ACCESS Interface to OLE DB supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.13  Features by Host Environment for OLE DB

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for OLE DB” on page 771.

SAS/ACCESS Interface to Oracle: Supported Features

Here are the features that SAS/ACCESS Interface to Oracle supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 10.14  Features by Host Environment for Oracle

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
## SAS/ACCESS Interface to PostgreSQL: Supported Features

Here are the features that SAS/ACCESS Interface to PostgreSQL supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

**Table 10.15 Features by Host Environment for PostgreSQL**

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for PostgreSQL” on page 820.
## SAS/ACCESS Interface to SAP HANA: Supported Features

Here are the features that SAS/ACCESS Interface to SAP HANA supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

### Table 10.16 Features by Host Environment for SAP HANA

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Bulk Loading for SAP HANA” on page 841.

## SAS/ACCESS Interface to Sybase: Supported Features

Here are the features that SAS/ACCESS Interface to Sybase supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

### Table 10.17 Features by Host Environment for Sybase

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and the “Bulk Loading for Sybase” on page 862.

### SAS/ACCESS Interface to Sybase IQ: Supported Features

Here are the features that SAS/ACCESS Interface to Sybase IQ supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

**Table 10.18  Features by Host Environment for Sybase IQ**

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Maximizing Teradata Load and Read Performance” on page 905.

SAS/ACCESS Interface to Teradata: Supported Features

Here are the features that SAS/ACCESS Interface to Teradata supports. To find out which versions of your DBMS are supported, see your system requirements documentation.

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>z/OS</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4 and “Maximizing Teradata Load and Read Performance” on page 905.

SAS/ACCESS Interface to Vertica: Supported Features

Here are the features that SAS/ACCESS Interface to Vertica supports. To find out which versions of your DBMS are supported, see your system requirements documentation.
<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>ACCESS Procedure</th>
<th>DBLOAD Procedure</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solaris for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows for x86</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows for x64</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For information about these features, see “Methods for Accessing Relational Database Data” on page 4.
Chapter 11
The LIBNAME Statement for Relational Databases

Overview: LIBNAME Statement for Relational Databases
Assigning Librefs .......................................................... 96
Sorting Data .......................................................................... 96
Using SAS Functions .......................................................... 96
Assigning a Libref Interactively ........................................... 97
Dictionary .............................................................................. 97
LIBNAME Statement Syntax for Relational Databases .............. 97
ACCESS= LIBNAME Option ................................................... 102
ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS= LIBNAME Option ...
ADJUST_NCHAR_COLUMN_LENGTHS= LIBNAME Option .............. 103
ALLOWED_SQLCODES= LIBNAME Option .................................. 105
AUTHDOMAIN= LIBNAME Option ............................................ 105
AUTHID= LIBNAME Option ................................................... 106
AUTOCOMMIT= LIBNAME Option ............................................ 107
BL_DEFAULT_DIR= LIBNAME Option ....................................... 107
BL_HOST= LIBNAME Option .................................................. 108
BL_KEEPIDENTITY= LIBNAME Option ..................................... 109
BL_KEEPNULLS= LIBNAME Option ........................................... 109
BL_LOG= LIBNAME Option .................................................... 110
BL_OPTIONS= LIBNAME Option .............................................. 110
BL_PORT= LIBNAME Option .................................................. 111
BULKLOAD= LIBNAME Option ............................................... 112
BULKUNLOAD= LIBNAME Option ........................................... 112
CAST= LIBNAME Option ...................................................... 113
CAST_OVERHEAD_MAXPERCENT= LIBNAME Option ..................... 115
CELLPROP= LIBNAME Option ................................................ 115
CHAR_AS_NCHAR= LIBNAME Option ....................................... 116
COMMAND_TIMEOUT= LIBNAME Option .................................... 117
CONFIG= LIBNAME Option .................................................... 117
CONFIGDIR= LIBNAME Option ............................................... 118
CONNECTION= LIBNAME Option .......................................... 118
CONNECTION_GROUP= LIBNAME Option .................................... 124
CURSOR_TYPE= LIBNAME Option ............................................ 125
DATETIME2= LIBNAME Option ............................................... 127
DB_LENGTH_SEMANTICS_BYTE= LIBNAME Option ....................... 128
DB_OBJECTS= LIBNAME Option ............................................. 129
DBCLIENT_ENCODING_FIXED= LIBNAME Option ......................... 129
DBCLIENT_MAX_BYTES= LIBNAME Option ................................ 131
DBCOMMIT= LIBNAME Option ............................................... 132
DBCINIT= LIBNAME Option .................................................. 133
DBCOUNTER= LIBNAME Option ........................................ 135
DBCREATE_TABLE_EXTERNAL= LIBNAME Option ................... 136
DBCREATE_TABLE_OPTS= LIBNAME Option ......................... 137
DBGEN_NAME= LIBNAME Option .................................... 138
DBINDEX= LIBNAME Option .......................................... 139
DBLIBINIT= LIBNAME Option ....................................... 140
DBLIBTERM= LIBNAME Option ....................................... 141
DBLINK= LIBNAME Option ........................................... 142
DBMAX_TEXT= LIBNAME Option ...................................... 142
DBMSTEMP= LIBNAME Option ........................................ 143
DBNULLKEYS= LIBNAME Option ...................................... 145
DBPROMPT= LIBNAME Option ......................................... 146
DBASASLABEL= LIBNAME Option .................................... 148
DBSERVER_ENCODING_FIXED= LIBNAME Option .................. 148
DBSERVER_MAX_BYTES= LIBNAME Option ......................... 150
DBSLICEPARAM= LIBNAME Option .................................. 151
DEFER= LIBNAME Option ............................................ 153
DEGREE= LIBNAME Option ........................................... 154
DELETE_MULT_ROWS= LIBNAME Option ............................ 155
DIMENSION= LIBNAME Option ...................................... 155
DIRECT_EXE= LIBNAME Option ..................................... 156
DIRECT_SQL= LIBNAME Option ...................................... 157
DRIVER_VENDOR= LIBNAME Option ................................ 159
ENABLE_BULK= LIBNAME Option .................................... 159
ERRLIMIT= LIBNAME Option ........................................ 160
ESCAPE_BACKSLASH= LIBNAME Option .............................. 161
FASTEXPORT= LIBNAME Option ..................................... 161
FASTLOAD= LIBNAME Option ........................................ 163
FETCH_IDENTITY= LIBNAME Option ................................ 164
HDFS_PRINCIPAL= LIBNAME Option ................................ 165
IGNORE_READ_ONLY_COLUMNS= LIBNAME Option .................. 165
IMPALA_PRINCIPAL= LIBNAME Option .............................. 167
IN= LIBNAME Option ................................................. 167
INSERT_SQL= LIBNAME Option ..................................... 168
INSERTBUFF= LIBNAME Option ..................................... 169
INTERFACE= LIBNAME Option ....................................... 170
KEYSET_SIZE= LIBNAME Option .................................... 170
LOCATION= LIBNAME Option ....................................... 171
LOCKTABLE= LIBNAME Option ...................................... 172
LOCKTIME= LIBNAME Option ........................................ 172
LOCKWAIT= LIBNAME Option ........................................ 173
LOGDB= LIBNAME Option ............................................ 173
LOGIN_TIMEOUT= LIBNAME Option .................................. 174
MAX_CONNECTS= LIBNAME Option .................................. 175
MODE= LIBNAME Option ............................................. 175
MULTI_DATASRC_OPTS= LIBNAME Option ......................... 176
MULTISTMT= LIBNAME Option ...................................... 178
OR_BINARY_DOUBLE= LIBNAME Option ............................. 178
OR_ENABLE_INTERRUPT= LIBNAME Option ......................... 179
OR_UPD_NOWHERE= LIBNAME Option ................................. 180
PACKETSIZE= LIBNAME Option ..................................... 180
PARMDDEFAULT= LIBNAME Option .................................. 181
PARMSTRING= LIBNAME Option ..................................... 181
PARTITION_KEY= LIBNAME Option .................................. 183
POST_STMT_OPTS= LIBNAME Option ................................. 184
PRESERVE_COL_NAMES= LIBNAME Option ......................... 185
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE_ISOLATION_LEVEL</td>
<td>LIBNAME Option</td>
<td>186</td>
</tr>
<tr>
<td>PRESERVE_GUID= LIBNAME Option</td>
<td></td>
<td>191</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES= LIBNAME Option</td>
<td></td>
<td>193</td>
</tr>
<tr>
<td>PRESERVE_USER= LIBNAME Option</td>
<td></td>
<td>194</td>
</tr>
<tr>
<td>PROGRAM_NAME= LIBNAME Option</td>
<td></td>
<td>195</td>
</tr>
<tr>
<td>PROPERTIES= LIBNAME Option</td>
<td></td>
<td>196</td>
</tr>
<tr>
<td>QUALIFIER= LIBNAME Option</td>
<td></td>
<td>197</td>
</tr>
<tr>
<td>QUALIFY_ROWS= LIBNAME Option</td>
<td></td>
<td>197</td>
</tr>
<tr>
<td>QUERY_BAND= LIBNAME Option</td>
<td></td>
<td>197</td>
</tr>
<tr>
<td>QUOTE_CHAR= LIBNAME Option</td>
<td></td>
<td>198</td>
</tr>
<tr>
<td>QUOTED_IDENTIFIER= LIBNAME Option</td>
<td></td>
<td>199</td>
</tr>
<tr>
<td>READBUFF= LIBNAME Option</td>
<td></td>
<td>200</td>
</tr>
<tr>
<td>READLOCK= LIBNAME Option</td>
<td></td>
<td>201</td>
</tr>
<tr>
<td>READ_MODE_WAIT= LIBNAME Option</td>
<td></td>
<td>202</td>
</tr>
<tr>
<td>REMOTE_DBTYPE= LIBNAME Option</td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>REREAD_EXPOSURE= LIBNAME Option</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>RESULTS= LIBNAME Option</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td>SAS_DBMS_AUTOMETADATA= LIBNAME Option</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>SCHEMA= LIBNAME Option</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>SESSIONS= LIBNAME Option</td>
<td></td>
<td>208</td>
</tr>
<tr>
<td>SHOW_SYNONYMS= LIBNAME Option</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td>SLEEP= LIBNAME Option</td>
<td></td>
<td>210</td>
</tr>
<tr>
<td>SPOOL= LIBNAME Option</td>
<td></td>
<td>211</td>
</tr>
<tr>
<td>SQL_FUNCTIONS= LIBNAME Option</td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY= LIBNAME Option</td>
<td></td>
<td>213</td>
</tr>
<tr>
<td>SQL_OJ_ANSI= LIBNAME Option</td>
<td></td>
<td>217</td>
</tr>
<tr>
<td>SQL_GENERATION= LIBNAME Option</td>
<td></td>
<td>218</td>
</tr>
<tr>
<td>STRINGDATES= LIBNAME Option</td>
<td></td>
<td>218</td>
</tr>
<tr>
<td>SYNONYMS= LIBNAME Option</td>
<td></td>
<td>219</td>
</tr>
<tr>
<td>TABLE_TYPE= LIBNAME Option</td>
<td></td>
<td>220</td>
</tr>
<tr>
<td>TENACITY= LIBNAME Option</td>
<td></td>
<td>221</td>
</tr>
<tr>
<td>TPT= LIBNAME Option</td>
<td></td>
<td>222</td>
</tr>
<tr>
<td>TPT_MAX_SESSIONS= LIBNAME Option</td>
<td></td>
<td>223</td>
</tr>
<tr>
<td>TPT_MIN_SESSIONS= LIBNAME Option</td>
<td></td>
<td>224</td>
</tr>
<tr>
<td>TR_ENABLE_INTERRUPT= LIBNAME Option</td>
<td></td>
<td>225</td>
</tr>
<tr>
<td>TRACE= LIBNAME Option</td>
<td></td>
<td>226</td>
</tr>
<tr>
<td>TRACEFILE= LIBNAME Option</td>
<td></td>
<td>227</td>
</tr>
<tr>
<td>TRANSCODE_FAIL= LIBNAME Option</td>
<td></td>
<td>227</td>
</tr>
<tr>
<td>UPDATE_ISO_OLATION_LEVEL= LIBNAME Option</td>
<td></td>
<td>228</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE= LIBNAME Option</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>UPDATE_MODE_WAIT= LIBNAME Option</td>
<td></td>
<td>230</td>
</tr>
<tr>
<td>UPDATE_MULTI_ROWS= LIBNAME Option</td>
<td></td>
<td>231</td>
</tr>
<tr>
<td>UPDATE_SQL= LIBNAME Option</td>
<td></td>
<td>231</td>
</tr>
<tr>
<td>UPDATEBUFF= LIBNAME Option</td>
<td></td>
<td>232</td>
</tr>
<tr>
<td>USE_DATADIRECT= LIBNAME Option</td>
<td></td>
<td>233</td>
</tr>
<tr>
<td>USE_ODBC_CL= LIBNAME Option</td>
<td></td>
<td>234</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT= LIBNAME Option</td>
<td></td>
<td>234</td>
</tr>
<tr>
<td>WARN_BIGINT= LIBNAME Option</td>
<td></td>
<td>234</td>
</tr>
</tbody>
</table>
Overview: LIBNAME Statement for Relational Databases

Assigning Librefs

The SAS/ACCESS LIBNAME statement extends the SAS global LIBNAME statement so that you can assign a libref to a relational DBMS. This feature lets you reference a DBMS object directly in a DATA step or SAS procedure. You can use it to read from and write to a DBMS object as if it were a SAS data set. You can associate a SAS libref with a relational DBMS database, schema, server, or group of tables and views.

For details about the syntax, see “LIBNAME Statement Syntax for Relational Databases” on page 97. For the engine name, connection options, and LIBNAME options for your SAS/ACCESS interface, see the DBMS-specific reference section for your SAS/ACCESS interface.

Sorting Data

When you use the SAS/ACCESS LIBNAME statement to associate a libref with relational DBMS data, you might observe some behavior that differs from that of normal SAS librefs. Because these librefs refer to database objects, such as tables and views, they are stored in the format of your DBMS. DBMS format differs from the format of normal SAS data sets. This is helpful to remember when you access and work with DBMS data.

For example, you can sort the observations in a normal SAS data set and store the output to another data set. However, in a relational DBMS, sorting data often has no effect on how it is stored. Because you cannot depend on your data to be sorted in the DBMS, you must sort the data at the time of query. Also, when you sort DBMS data, results might vary depending on whether your DBMS places data with NULL values (which are translated in SAS to missing values) at the beginning or the end of the result set.

Using SAS Functions

When you use librefs that refer to DBMS data with SAS functions, some functions might return a value that differs from what is returned when you use the functions with normal SAS data sets. For example, the PATHNAME function might return a blank value. For a normal SAS libref, a blank value means that the libref is not valid. However, for a libref associated with a DBMS object, a blank value means only that there is no pathname associated with the libref.

Usage of some functions might also vary. For example, the LIBNAME function can accept an optional SAS-library argument. When you use the LIBNAME function to assign or unassign a libref that refers to DBMS data, you omit this argument. For full details about how to use SAS functions, see the SAS Functions and CALL Routines: Reference.
Assigning a Libref Interactively

An easy way to associate a libref with a relational DBMS is to use the New Library window. One method to open this window is to issue the DMLIBASSIGN command from your SAS session's command box or command line. You can also open the window by clicking the file cabinet icon in the SAS Explorer toolbar. Below, the user Samantha assigns a libref MYORADB to an Oracle database that the SQL*Net alias ORAHRDEPT references. By using the SCHEMA= LIBNAME option, Samantha can access database objects that another user owns.

Figure 11.1 New Library Window

Here is how to use the features of the New Library window.

- **Name**: Enter the libref that you want to assign to a SAS library or a relational DBMS.
- **Engine**: Click the down arrow to select a name from the pull-down listing.
- **Enable at startup**: Click this if you want the specified libref to be assigned automatically when you open a SAS session.
- **Library Information**: These fields represent the SAS/ACCESS connection options and vary according to the SAS/ACCESS engine that you specify. Enter the appropriate information for your site's DBMS. The Options field lets you enter SAS/ACCESS LIBNAME options. Use blanks to separate multiple options.
- **OK**: Click this button to assign the libref, or click **Cancel** to exit the window without assigning a libref.

Dictionary

**LIBNAME Statement Syntax for Relational Databases**

Associates a SAS libref with a DBMS database, schema, server, or a group of tables and views.

**Valid in**: anywhere
See:  
“Overview: LIBNAME Statement for Relational Databases”

Syntax

Form 1:  
LIBNAME libref engine-name
<\SAS/ACCESS-connection-options>
<\SAS/ACCESS-LIBNAME-options>;

Form 2:  
LIBNAME libref CLEAR | _ALL_ CLEAR;

Form 3:  
LIBNAME libref LIST | _ALL_ LIST;

Required Arguments

libref
specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views. Like the global SAS LIBNAME statement, the SAS/ACCESS LIBNAME statement creates shortcuts or nicknames for data storage locations. A SAS libref is an alias for a virtual or physical directory. A SAS/ACCESS libref is an alias for the DBMS database, schema, or server where your tables and views are stored.

database engine-name
specifies the SAS/ACCESS engine name for your DBMS, such as oracle or db2. The engine name is required. Because the SAS/ACCESS LIBNAME statement associates a libref with a SAS/ACCESS engine that supports connections to a particular DBMS, it requires a DBMS-specific engine name. See the DBMS-specific reference section for details.

CLEAR

disassociates one or more currently assigned librefs.
Specify libref to disassociate a single libref. Specify _ALL_ to disassociate all currently assigned librefs.

_ALL_

specifies that the CLEAR or LIST argument applies to all currently assigned librefs.

LIST

writes the attributes of one or more SAS/ACCESS libraries or SAS libraries to the SAS log.
Specify libref to list the attributes of a single SAS/ACCESS library or SAS library. Specify _ALL_ to list the attributes of all libraries that have librefs in your current session.

Optional Arguments

SAS/ACCESS-connection-options
provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS; these arguments are different for each database. For example, to connect to an Oracle database, your connection options are USER=, PASSWORD=, and PATH=:

libname myoralib oracle user=myusr1 password=mypwd1 path='mysrv1';

If the connection options contain characters that are not allowed in SAS names, enclose the values of the arguments in quotation marks. On some DBMSs, if you specify the appropriate system options or environment variables for your database,
you can omit the connection options. For connection option details, see the DBMS-specific information for your SAS/ACCESS interface.

**SAS/ACCESS-LIBNAME-options**

define how DBMS objects are processed by SAS. Some LIBNAME options can enhance performance; others determine locking or naming behavior. For example, the PRESERVE_COL_NAMES= option lets you specify whether to preserve spaces, special characters, and mixed case in DBMS column names when creating tables. The availability and default behavior of many of these options are DBMS-specific. See the DBMS-specific reference section for LIBNAME options that are available for your SAS/ACCESS interface. For general information, see LIBNAME Options for Relational Databases on page 101.

### Details

**Form 1: Using Data from a DBMS**

You can use a LIBNAME statement to read from and write to a DBMS table or view as if it were a SAS data set.

For example, in MYDBLIB.EMPLOYEES_Q2, MYDBLIB is a SAS libref that points to a particular group of DBMS objects, and EMPLOYEES_Q2 is a DBMS table name. When you specify MYDBLIB.EMPLOYEES_Q2 in a DATA step or procedure, you dynamically access the DBMS table. SAS supports reading, updating, creating, and deleting DBMS tables dynamically.

**Form 2: Disassociating a Libref from a SAS Library**

To disassociate or clear a libref from a DBMS, use a LIBNAME statement. Specify the libref (for example, MYDBLIB) and the CLEAR option as shown here:

```plaintext
libname mydblib CLEAR;
```

You can clear a single specified libref or all current librefs.

The database engine disconnects from the database and closes any free threads or resources that are associated with that libref’s connection.

**Form 3: Writing SAS Library Attributes to the SAS Log**

Use a LIBNAME statement to write the attributes of one or more SAS/ACCESS libraries or SAS libraries to the SAS log. Specify libref to list the attributes of a single SAS/ACCESS library or SAS library, as shown below.

```plaintext
libname mydblib LIST;
```

Specify _ALL_ to list the attributes of all libraries that have librefs in your current session.

**SQL Views with Embedded LIBNAME Statements**

With SAS software, you can embed LIBNAME statements in the definition of an SQL view. This means that you can store a LIBNAME statement in an SQL view that contains all information that is required to connect to a DBMS. Whenever the SQL view is read, PROC SQL uses the embedded LIBNAME statement to assign a libref. After the view has been processed, PROC SQL unassigns the libref.

In this example, an SQL view of the Oracle table DEPT is created. Whenever you use this view in a SAS program, the ORALIB library is assigned. The library uses the connection information (user name, password, and data source) that is provided in the embedded LIBNAME statement.
proc sql;
  create view sasuser.myview as
  select dname from oralib.dept
  using libname oralib oracle
    user=scott pw=tiger datasrc=orsrv;
quit;

Note: You can use the USING LIBNAME syntax to embed LIBNAME statements in SQL views. For more information about the USING LIBNAME syntax, see the SAS SQL Procedure User's Guide.

**Assigning a Libref with a SAS/ACCESS LIBNAME Statement**

This statement creates a libref, MYDBLIB, that uses the SAS/ACCESS interface to DB2.

libname mydblib db2 ssid=db2a authid=testid server=os390svr;

The statement below associates the SAS libref MYDBLIB with an Oracle database that uses the SQL*Net alias AIRDB_REMOTE. You specify the SCHEMA= option in the SAS/ACCESS LIBNAME statement to connect to the Oracle schema in which the database resides. In this example, Oracle schemas reside in a database.

libname mydblib oracle user=myusr1 password=mypwd1
  path=airdb_remote schema=hrdept;

The AIRDB_REMOTE database contains a number of DBMS objects, including several tables, such as STAFF. After you assign the libref, you can reference the Oracle table like a SAS data set and use it as a data source in any DATA step or SAS procedure. In the SQL procedure statement below, MYDBLIB.STAFF is the two-level SAS name for the STAFF table in the Oracle database AIRDB_REMOTE.

proc sql;
  select idnum, lname
  from mydblib.staff
  where state='NY'
  order by lname;
quit;

You can use the DBMS data to create a SAS data set.

data newds;
  set mydblib.staff(keep=idnum lname fname);
run;

You can also use the libref and data set with any other SAS procedure. This statement prints the information in the STAFF table.

proc print data=mydblib.staff;
run;

This statement lists the database objects in the MYDBLIB library.

proc datasets library=mydblib;
quit;

**Using the Prompting Window When Specifying LIBNAME Options**

This statement uses the DBPROMPT= LIBNAME option to cause the DBMS connection prompting window to appear and prompt you for connection information.

libname mydblib oracle dbprompt=yes;
When you use this option, you enter connection information into the fields in the prompting window rather than in the LIBNAME statement.

You can add the DEFER= NO LIBNAME option to make the prompting window appear when the libref is assigned rather than when the table is opened.

```sas
libname mydblib oracle dbprompt=yes defer=no;
```

**Assigning a Libref to a Remote DBMS**

SAS/CONNECT (single-user) and SAS/SHARE (multiple user) software give you access to data by means of remote library services (RLS). RLS lets you access your data on a remote machine as if it were local. For example, it permits a graphical interface to reside on the local machine. The data remains on the remote machine.

This access is given to data stored in many types of SAS files. Examples include external databases (through the SAS/ACCESS LIBNAME statement and views that are created with it) and SAS data views (views that are created with PROC SQL, the DATA step, and SAS/ACCESS software). RLS lets you access SAS data sets, SAS views, and relational DBMS data that SAS/ACCESS LIBNAME statements define. For more information, see the discussion about remote library services in the SAS/SHARE User's Guide.

You can use RLS to update relational DBMS tables that are referenced with the SAS/ACCESS LIBNAME statement.

In the next example, the SAS/ACCESS LIBNAME statement makes a connection to a DB2 database that resides on the remote SAS/SHARE server REMOS390. This LIBNAME statement is submitted in a local SAS session. The SAS/ACCESS engine name is specified in the remote option RENGINE=. The DB2 connection option and any LIBNAME options are specified in the remote option ROPTIONS=. Options are separated by a blank space. RLSDB2.EMPLOYEES is a SAS data set that references the DB2 table EMPLOYEES.

```sas
libname rlsdb2 rengine=db2 server=remos390
   roptions="ssid=db2a authid=testid";
proc print data=rlsdb2.employees;
run;
```

**LIBNAME Options for Relational Databases**

When you specify an option in the LIBNAME statement, it applies to all objects (such as tables and views) in the database that the libref represents. For information about options that you specify on individual SAS data sets, see “About the Data Set Options for Relational Databases” on page 241. For general information, see “LIBNAME Statement Syntax for Relational Databases” on page 97. See the DBMS-specific reference section for LIBNAME options that are available for your SAS/ACCESS interface.

Many LIBNAME options are also available for use with the SQL pass-through facility. See the section on the SQL pass-through facility in the documentation for your SAS/ACCESS interface to determine which LIBNAME options are available in the SQL pass-through facility for your DBMS. For general information, see “SQL Pass-Through Facility” on page 471.

When a like-named option is specified in both the LIBNAME statement and after a data set name, SAS uses the value that is specified on the data set name.
ACCESS= LIBNAME Option
Determines the access level with which a libref connection is opened.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

Syntax
ACCESS=READONLY

Syntax Description
READONLY specifies that you can read but not update tables and views.

Details
Using this option prevents writing to the DBMS. If this option is omitted, you can read and update tables and views if you have the necessary DBMS privileges.

ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS= LIBNAME Option
Specifies whether to adjust the lengths of CHAR or VARCHAR data type columns with byte semantic column lengths.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: based on the value of DBCLIENT_MAX_BYTES=.
Interactions: When the DBCLIENT_MAX_BYTES= value is greater than 1, ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES. When DBCLIENT_MAX_BYTES=1, ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=NO.
When ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES and DBSERVER_ENCODING_FIXED=YES, this changes the character length calculation. (See “Details”.)

Supports: NLS
Data source: Oracle
See: ADJUST_NCHAR_COLUMN_LENGTHS= LIBNAME option, DBCLIENT_MAX_BYTES= LIBNAME option, DBSERVER_ENCODING_FIXED= LIBNAME option on page 148, DBSERVER_MAX_BYTES= LIBNAME option
Syntax

`ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS= YES | NO`

Syntax Description

**YES**
indicates that column lengths are based on the number of characters multiplied by the `DBCLIENT_MAX_BYTES=` value. When the `DBCLIENT_MAX_BYTES=` value is greater than 1, `ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES`.

**NO**
indicates that any column lengths that byte semantics specify on the server are used “as is” on the client. If `DBCLIENT_MAX_BYTES=1`, `ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=NO`.

Details

When `ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES` and `DBSERVER_ENCODING_FIXED=YES`, character length is calculated as byte length divided by the `DBSERVER_MAX_BYTES=` value. When `ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES` and `DBSERVER_ENCODING_FIXED=NO`, character lengths are not adjusted.

Example: Adjust Client-Encoded Column Lengths

When `ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES`, column lengths that byte semantics creates are adjusted with client encoding, as shown in this example.

```plaintext
libname x3 &engine &connopt ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES;
proc contents data=x3.char_sem; run;
proc contents data=x3.nchar_sem; run;
proc contents data=x3.byte_sem; run;
proc contents data=x3.mixed_sem; run;
```

In this example, various options have different settings.

```plaintext
libname x5 &engine &connopt ADJUST_NCHAR_COLUMN_LENGTHS=NO
ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=NO DBCLIENT_MAX_BYTES=3;
proc contents data=x5.char_sem; run;
proc contents data=x5.nchar_sem; run;
proc contents data=x5.byte_sem; run;
proc contents data=x5.mixed_sem; run;
```

This example also uses different settings for the various options.

```plaintext
libname x6 &engine &connopt ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES
ADJUST_NCHAR_COLUMN_LENGTHS=YES DBCLIENT_MAX_BYTES=3;
proc contents data=x6.char_sem; run;
proc contents data=x6.nchar_sem; run;
proc contents data=x6.byte_sem; run;
proc contents data=x6.mixed_sem; run;
```
Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: YES
Supports: NLS
Data source: Oracle
See: ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS= LIBNAME option, DBCLIENT_MAX_BYTES= LIBNAME option

Syntax

ADJUST_NCHAR_COLUMN_LENGTHS=YES | NO

Syntax Description

YES
- indicates that column lengths are based on the number of characters multiplied by the DBCLIENT_MAX_BYTES= value.

NO
- indicates that column lengths that NCHAR or VARCHAR columns specify are multiplied by 2.

Example: No Adjustment for Client-Encoded Column Lengths

NCHAR column lengths are no longer adjusted to client encoding when ADJUST_NCHAR_COLUMN_LENGTHS=NO, as shown in this example.

```
libname x2 &engine &connopt ADJUST_NCHAR_COLUMN_LENGTHS=NO;
proc contents data=x2.char_sem; run;
proc contents data=x2.nchar_sem; run;
proc contents data=x2.byte_sem; run;
proc contents data=x2.mixed_sem; run;
```

In this example, various options have different settings.

```
libname x5 &engine &connopt ADJUST_NCHAR_COLUMN_LENGTHS=NO
ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=NO DBCLIENT_MAX_BYTES=3;
proc contents data=x5.char_sem; run;
proc contents data=x5.nchar_sem; run;
proc contents data=x5.byte_sem; run;
proc contents data=x5.mixed_sem; run;
```

This example also uses different settings for the various options.

```
libname x6 &engine &connopt ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES
ADJUST_NCHAR_COLUMN_LENGTHS=YES DBCLIENT_MAX_BYTES=3;
proc contents data=x6.char_sem; run;
proc contents data=x6.nchar_sem; run;
proc contents data=x6.byteSem; run;
proc contents data=x6.mixed_sem; run;
```
ALLOWED_SQLCODES= LIBNAME Option

Specifies the SQL warnings and errors to ignore during preparation and execution.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Requirement: You must separate multiple values with commas and enclose the entire list in parentheses. Use double quotation marks around each value in the list.
Data source: DB2 under z/OS
Note: Support for this LIBNAME option was added in the first maintenance release for SAS 9.4.
Example: libname x db2 allowed_sqlcodes=('144','-803') ssid=DB2;

Syntax
ALLOWED_SQLCODES=(sqlcode<, sqlcode<, sqlcode …>>)

Syntax Description
sqlcode
specifies one or more values that you want to control. These values subsequently do not appear in the log.

Details
Use this option to pass a list of specific warnings and errors for the SAS/ACCESS engine to ignore. This is helpful to allow execution to continue for errors or warnings that are for informational purposes only.

AUTHDOMAIN= LIBNAME Option

Allows connection to a server by specifying the name of an authentication domain metadata object.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Requirements: For data sources with the SERVER= option, if you specify AUTHDOMAIN=, you must also specify the METASERVER= system option. However, the authentication domain references credentials, so you do not need to explicitly specify database USER= and PASSWORD=. Here is an example:

options metauser="metadata-userid" metapass="metadata-password" metaport=8561 metaprotocol=bridge metarepository="metadata-repository" metaserver="server-name";

libname A1 saphana server=mysrv1 port=30015 authodomain="hanaauth";

The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running to resolve the metadata object specification.
Interaction: To specify AUTHDOMAIN=, you must also specify SERVER=.

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: For complete information about creating and using authentication domains, see the credential management topic in SAS Intelligence Platform: Security Administration Guide.

Syntax

AUTHDOMAIN=authentication-domain

Syntax Description

authentication-domain
specifies the name of an authentication domain metadata object.

Details

An administrator creates authentication domain definitions while creating a user definition with the User Manager in SAS Management Console. The authentication domain is associated with one or more login metadata objects, which provide access to the server. The authentication domain is resolved when the DBMS engine calls the SAS Metadata Server and returns the authentication credentials.

AUTHID= LIBNAME Option

Allows qualified table names with an authorization ID, a user ID, or a group ID.

Valid in: SAS/ACCESS LIBNAME statement
Alias: SCHEMA=
Default: none
Data source: DB2 under z/OS
See: AUTHID= data set option

Syntax

AUTHID=authorization-ID

Syntax Description

authorization-ID
cannot exceed eight characters.

Details

When you specify the AUTHID= option, every table that is referenced by the libref is qualified as authidtablename before any SQL code is passed to the DBMS. If you do not specify a value for AUTHID=, the table name is not qualified before it is passed to
the DBMS. After the DBMS receives the table name, it automatically qualifies it with your user ID. You can override the LIBNAME AUTHID= option by using the AUTHID= data set option. This option is not validated until you access a table.

**AUTOCOMMIT= LIBNAME Option**

Indicates whether updates are committed immediately after they are submitted.

**Valid in:** SAS/ACCESS LIBNAME statement [all], CONNECT statement [Netezza]

**Defaults:** operation-specific [Aster, Greenplum, HAWQ, Sybase IQ]

varies with the transaction type [DB2 under UNIX and PC Hosts, Microsoft SQL Server]

data source-specific [ODBC, OLE DB]

NO [if you are using the SAS/ACCESS LIBNAME statement, the data source provider supports transactions, and the connection is to update data]

NO [for updates and the main LIBNAME connection for Netezza]

NO [PostgreSQL, SAP HANA, Vertica]

YES [for PROC PRINT, Read-only connections, and the SQL pass-through facility]

YES [Informix, MySQL, Sybase]

**Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** AUTOCOMMIT= data set option

**Syntax**

AUTOCOMMIT=**YES | NO**

**Syntax Description**

**YES**

specifies that all updates, deletes, and inserts are committed (that is, saved to a table) immediately after they are submitted, and no rollback is possible.

**NO**

specifies that the SAS/ACCESS engine automatically performs the commit when it reaches the DBCOMMIT= value, or the default number of rows if DBCOMMIT is not set.

**BL_DEFAULT_DIR= LIBNAME Option**

Specifies where bulk loading creates all intermediate files.

**Valid in:** SAS/ACCESS LIBNAME statement

**Defaults:**

<current-directory> (MySQL)

<database-name> (Oracle)

**Restriction:** The value that you specify must not contain a space.
Requirements: To specify this option, you must first set BULKLOAD=YES. This option must end with a slash on Windows or a backslash on UNIX—for example:

\texttt{BL\_DEFAULT\_DIR='c:\temp'}
\texttt{BL\_DEFAULT\_DIR='/temp/'}

Supports: CTL, DAT, LOG, BAD, and DSC intermediate bulk-load files (Oracle)

Data source: MySQL, Oracle, PostgreSQL

See: BULKLOAD= LIBNAME option BL\_DATASET\_DIR= data set option

**Syntax**

\texttt{BL\_DEFAULT\_DIR='host-specific-directory-path'}

**Required Argument**

\textit{host-specific-directory-path}

specifies the host-specific directory path where intermediate bulk-loading files are created.

**Details**

The value that you specify for this option is prepended to the filename. Be sure to provide the complete, host-specific directory path that includes the file and directory separator character to accommodate all platforms.

**Example: Create All Files in a Temporary Directory**

In this example, bulk loading Oracle data creates all related files in the c:\temp directory:

\begin{verbatim}
libname mypath oracle user=myusr1 password=mypwd1
bulkload=yes bl_default_dir='c:\temp';
\end{verbatim}

**BL\_HOST= LIBNAME Option**

Specifies the WebHDFS host name or IP address of the server where the external data file is stored.

**Valid in:** SAS/ACCESS LIBNAME statement

**Alias:** BLHOST=, BULKLOAD\_HOST=

**Default:** SERVER= value, if SERVER= is set; otherwise, none.

**Restriction:** Enclose the name in quotation marks.

**Interaction:** You can set this option only when the BULKLOAD=LIBNAME option is set to 1.

**Data source:** Impala

**Note:** Support for Impala was added in the second maintenance release for SAS 9.4.

**See:** BULKLOAD= option
Syntax

BL_HOST='host-name'

**Syntax Description**

*host-name*

specifies the name or IP address of the server where the external data file is stored.

**Details**

When you are bulk loading data, use this option to specify the name or IP address of the server where external data is stored. By default, this value is the same as the value of SERVER=, when the SERVER= option is set.

---

**BL_KEEPIDENTITY= LIBNAME Option**

Determines whether the identity column that is created during bulk loading is populated with values that Microsoft SQL Server generates or with values that the user provides.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** NO
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** OLE DB
- **See:** BL_KEEPIDENTITY= data set option, BULKLOAD= LIBNAME option

**Syntax**

BL_KEEPIDENTITY=YES | NO

**Syntax Description**

**YES**

specifies that the user must provide values for the identity column.

**NO**

specifies that Microsoft SQL Server generates values for an identity column in the table.

**Details**

This option is valid only when you use the Microsoft SQL Server provider.

---

**BL_KEEPNULLS= LIBNAME Option**

Indicates how NULL values in Microsoft SQL Server columns that accept NULL are handled during bulk loading.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** YES
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
Data source: OLE DB
See: `BL_KEEPNULLS=` data set option, BULKLOAD= LIBNAME option

### Syntax

```
BL_KEEPNULLS=YES | NO
```

### Syntax Description

**YES**
- specifies that Microsoft SQL Server preserves NULL values that the OLE DB interface inserts.

**NO**
- specifies that Microsoft SQL Server replaces NULL values that the OLE DB interface inserts with a default value, as specified in the DEFAULT constraint.

### Details

This option affects values in only Microsoft SQL Server columns that accept NULL and have a DEFAULT constraint.

---

### BL_LOG= LIBNAME Option

Specifies the name of the error file where all errors are written when BULKLOAD=YES.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>ODBC</td>
</tr>
<tr>
<td>Tip:</td>
<td>If you do not specify BL_LOG=, then errors are not recorded during bulk loading.</td>
</tr>
<tr>
<td>See:</td>
<td><code>BL_LOG=</code> data set option, BULKLOAD= LIBNAME option</td>
</tr>
</tbody>
</table>

### Syntax

```
BL_LOG='filename'
```

### Required Argument

- **filename**
  - specifies the name of the designated log file.

---

### BL_OPTIONS= LIBNAME Option

Passes options to the DBMS bulk-load facility, which affects how it loads and processes data.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>not specified</td>
</tr>
</tbody>
</table>

---
# Requirement
To specify this option, you must first set BULKLOAD=YES.

# Data source
ODBC, OLE DB

# See
BL_OPTIONS= data set option, BULKLOAD= LIBNAME option, BULKLOAD= data set option, UPDATE_LOCK_TYPE= LIBNAME option

## Syntax

$$BL\_OPTIONS='option <..., option>$$

### Required Argument

*option*

specifies a bulk-load option that you want to set.

## Details

You can use BL_OPTIONS= to pass options to the DBMS bulk-load facility when it is called, thereby affecting how data is loaded and processed. You must enclose the entire string of one or more options in quotation marks. Separate multiple options with commas.

By default, no options are specified. This option takes the same values as the -h HINT option of the Microsoft BCP utility. See the Microsoft SQL Server documentation for more information about bulk copy options.

This option is valid only when you use the Microsoft SQL Server driver or the Microsoft SQL Server provider on Windows platforms.

*ODBC*: Supported hints are ORDER, ROWS_PER_BATCH, KILOBYTES_PER_BATCH, TABLOCK, and CHECK_CONSTRAINTS. If you specify UPDATE_LOCK_TYPE=TABLE, the TABLOCK hint is automatically added.

---

## BL_PORT= LIBNAME Option

Specifies the port number for writing table data to the Hadoop cluster.

### Valid in:
SAS/ACCESS LIBNAME statement

### Alias:
BLPORT=, BULKLOAD_PORT=

### Default:
8020 [Hadoop], 50070 [Impala]

### Restriction:
This option is required if Hadoop HDFS service is running on a port other than 8020 (Hadoop) or 50070 (Impala).

### Data source:
Hadoop, Impala

### Note:
Support for Impala was added in the second maintenance release for SAS 9.4.

### Tip:
You can use this option without setting BULKLOAD=YES.

### See:
PORT= connection option

## Syntax

$$BL\_PORT=port$$
**Syntax Description**

**port**

specifies the port number to use to write table data.

**Details**

Use this option to specify only the port number that bulk loading uses to write data to the Hadoop cluster using the HDFS streaming server.

---

**BULKLOAD= LIBNAME Option**

Determines whether SAS uses a DBMS facility to insert data into a DBMS table.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Alias:** FASTLOAD= [Teradata]
- **Default:** NO

**Requirement:** Before you set BULKLOAD=YES, you must have set the SAS_HADOOP_RESTFUL environment variable to 1 before starting your SAS session. For more information, see *SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS*.

**Data source:** Hadoop, Impala, ODBC, OLE DB, Teradata

**Note:** Support for Impala was added in the second maintenance release for SAS 9.4.

**See:** BULKUNLOAD= LIBNAME option, BULKLOAD= data set option, BULKUNLOAD= data set option

---

**Syntax**

**BULKLOAD=** **YES | NO**

**Syntax Description**

**YES**

calls a DBMS-specific bulk-load facility to insert or append rows to a DBMS table.

**NO**

does not call the DBMS bulk-load facility.

---

**BULKUNLOAD= LIBNAME Option**

Rapidly retrieves (fetches) a large number of rows from a data set.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** NO

**Restriction:** To set BULKUNLOAD=YES for Netezza, you must be the Netezza admin user, or you must have Create External Table permission.

**Data source:** Aster, Netezza

**Note:** Support for Aster was added in the second maintenance release for SAS 9.4.
**BULKUNLOAD= data set option, BULKLOAD= LIBNAME option, Netezza bulk unloading, Aster bulk unloading**

**Syntax**

```plaintext
BULKUNLOAD=YES | NO
```

**Syntax Description**

**YES**
calls the Remote External Table interface to retrieve data from the server.

**NO**
uses standard result sets to retrieve data from the DBMS.

**Details**

Using BULKUNLOAD=YES is the fastest way to retrieve large numbers of rows from a Netezza table.

---

**CAST= LIBNAME Option**

Specifies whether SAS or the Teradata DBMS server performs data conversions.

**Valid in:** SAS/ACCESS LIBNAME statement

**Default:** none

**Data source:** Teradata

**See:** CAST= data set option, CAST_OVERHEAD_MAXPERCENT= LIBNAME option, CAST_OVERHEAD_MAXPERCENT= data set option

**Syntax**

```plaintext
CAST=YES | NO
```

**Syntax Description**

**YES**
forces data conversions (casting) to be done on the Teradata DBMS server and overrides any data overhead percentage limit.

**NO**
forces data conversions to be done by SAS and overrides any data overhead percentage limit.

**Details**

Internally, SAS numbers and dates are floating-point values. Teradata has varying formats for numbers, including integers, floating-point values, and decimal values. Number conversion must occur when you are reading Teradata numbers that are not floating point (Teradata FLOAT). SAS/ACCESS can use the Teradata CAST= function to cause Teradata to perform numeric conversions. The parallelism of Teradata makes it suitable for performing this work. This is especially true when running SAS on z/OS, where CPU activity can be costly.
CAST= can cause more data to be transferred from Teradata to SAS, as a result of the option forcing the Teradata type into a larger SAS type. For example, the CAST= transfer of a Teradata BYTEINT to SAS floating point adds seven overhead bytes to each row transferred.

These Teradata types are candidates for casting.

- INTEGER
- BYTEINT
- SMALLINT
- DECIMAL
- DATE

SAS/ACCESS limits data expansion for CAST= to 20% to trade rapid data conversion by Teradata for extra data transmission. If casting does not exceed a 20% data increase, all candidate columns are cast. If the increase exceeds this limit, SAS attempts to cast only Teradata DECIMAL types. If casting only DECIMAL types still exceeds the increase limit, SAS performs the data conversions.

You can alter the casting rules by using the CAST= or CAST_OVERHEAD_MAXPERCENT= LIBNAME option. With CAST_OVERHEAD_MAXPERCENT=, you can change the 20% overhead limit. With CAST=, you can override the percentage rules.

- CAST=YES forces Teradata to cast all candidate columns.
- CAST=NO cancels all Teradata casting.

CAST= applies only when you are reading Teradata tables into SAS, not when you are writing Teradata tables from SAS.

Also, CAST= applies only to SQL that SAS generates for you. If you supply your own SQL with the explicit SQL feature of PROC SQL, you must code your own casting clauses. Data conversions are therefore forced to occur in Teradata instead of SAS.

**Examples**

**Example 1: Force Casting for All Tables**

This example demonstrates the use of the CAST= option in a LIBNAME statement to force casting for all referenced tables.

```sas
libname mydblib teradata user=myusr1 pw=mypwd1 cast=yes;
proc print data=mydblib.emp;
  where empno<1000;
run;
proc print data=mydblib.sal;
  where salary>50000;
run;
```

**Example 2: Turn Casting Off for a Specific Table Reference**

This example demonstrates the use of the CAST= option in a table reference to turn casting off for that table.

```sas
proc print data=mydblib.emp (cast=no);
  where empno<1000;
run;
```
CAST_OVERHEAD_MAXPERCENT= LIBNAME Option

Specifies the overhead limit for data conversions to perform in Teradata instead of SAS.

Valid in: SAS/ACCESS LIBNAME statement
Default: 20
Data source: Teradata
See: CAST= LIBNAME option, CAST= data set option, CAST_OVERHEAD_MAXPERCENT= data set option

Syntax

CAST_OVERHEAD_MAXPERCENT=n

Syntax Description

n any positive numeric value. The engine default is 20.

Details

Teradata INTEGER, BYTEINT, SMALLINT, and DATE columns require conversion when read in to SAS. Either Teradata or SAS can perform conversions. When Teradata performs the conversion, the row size that is transmitted to SAS using the Teradata CAST operator can increase. CAST_OVERHEAD_MAXPERCENT= limits the allowable increase, also called conversion overhead.

Example: Increase the Allowable Overhead

This example demonstrates the use of CAST_OVERHEAD_MAXPERCENT= to increase the allowable overhead to 40%.

```sas
proc print data=mydblib.emp (cast_overhead_maxpercent=40);
where empno<1000;
run;
```

CELLPROP= LIBNAME Option

Modifies the metadata and content of a result data set that the MDX command defines.

Valid in: SAS/ACCESS LIBNAME statement
Default: VALUE
Data source: OLE DB
See: "Accessing OLE DB for OLAP Data"

Syntax

CELLPROP=VALUE | FORMATTED_VALUE
Syntax Description

VALUE
specifies that the SAS/ACCESS engine tries to return actual data values. If all values in a column are numeric, that column is defined as NUMERIC.

FORMATTED_VALUE
specifies that the SAS/ACCESS engine returns formatted data values. All columns are defined as CHARACTER.

Details

When an MDX command is issued, the resulting data set might have columns that contain one or more types of data values: the actual value of the cell or the formatted value of the cell.

For example, if you issue an MDX command and the resulting data set contains a column named SALARY, the column could contain data values of two types. It could contain numeric values, such as 50000, or it could contain formatted values, such as $50,000. Setting the CELLPROP= option determines how the values are defined and the value of the column.

It is possible for a column in a result set to contain both NUMERIC and CHARACTER data values. For example, a data set might return the data values of 50000, 60000, and UNKNOWN. SAS data sets cannot contain both types of data. In this situation, even if you specify CELLPROP=VALUE, the SAS/ACCESS engine defines the column as CHARACTER and returns formatted values for that column.

CHAR_AS_NCHAR= LIBNAME Option

Specifies the default character type to use for table columns.

Valid in: SAS/ACCESS LIBNAME statement
Default: NO
Data source: Netezza, SAP HANA
See: DBTYPE= data set option

Syntax

CHAR_AS_NCHAR=YES | NO

Syntax Description

YES
specifies that NCHAR or NVARCHAR be used as the default column type.

NO
specifies that CHAR or VARCHAR be used as the default column type.

Details

Use this option when you cannot use the DBTYPE= data set option on page 351 for table columns that contain multilingual character data.
Example: Set Multilingual Data as the Default

The SAS data set, local_cust, contains multilingual data in this example.

```sas
libname net netezza server=mysrv1 database=mydb1 uid=myusr1 pwd=mypwd1 CHAR_AS_NCHAR=YES;

data net.customers;
  set sas.local_cust;
run;
```

### COMMAND_TIMEOUT= LIBNAME Option

Specifies the number of seconds to wait before a data source command times out.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** 0
- **Data source:** OLE DB
- **See:** COMMAND_TIMEOUT= data set option

#### Syntax

```
COMMAND_TIMEOUT=number-of-seconds
```

#### Syntax Description

- **number-of-seconds**
  - an integer greater than or equal to 0, where 0 represents no time-out.

### CONFIG= LIBNAME Option

Specifies a path to the XML configuration file to use for bulk-load options.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Alias:** CFG=, HD_CONFIG=
- **Default:** none (SAS_HADOOP_CONFIG_PATH is the default when you set only BULKLOAD=YES)
- **Requirement:** You must enclose the value in quotation marks.
- **Data source:** Hadoop, Impala
- **Note:** Support for this option was added in the third maintenance release for SAS 9.4.
- **See:** BULKLOAD= LIBNAME option, CONFIG= data set option, CONFIGDIR= LIBNAME option, CONFIGDIR= data set option

#### Syntax

```
CONFIG='config-file'
```
**Required Argument**

`config-file`

specifies the path for the required configuration file for bulk-load options. This single XML file must contain information about the host, port, and Kerberos principal for HDFS.

---

**CONFIGDIR= LIBNAME Option**

Specifies a directory path to search for the required XML configuration files to use for bulk-load options.

**Valid in:** SAS/ACCESS LIBNAME statement

**Alias:** CFGDIR=, HD_CONFIGDIR=

**Default:** none (SAS_HADOOP_JAR_PATH is the default when you set only BULKLOAD=YES)

**Requirement:** You must enclose the value in quotation marks.

**Data source:** Hadoop, Impala

**Note:** Support for this option was added in the third maintenance release for SAS 9.4.

**See:** BULKLOAD= LIBNAME option, CONFIG= data set option, CONFIG= LIBNAME option, CONFIGDIR= data set option

---

**Syntax**

`CONFIGDIR='config-dir'`

**Required Argument**

`config-dir`

specifies the directory path to search for the required configuration files (core-site.xml and hdfs-site.xml) for bulk-load options that contain the required host, port, and Kerberos principal for HDFS.

---

**CONNECTION= LIBNAME Option**

Specifies whether operations on a single or multiple librefs can share a connection to the DBMS.

**Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement

**Default:** DBMS-specific

**Interaction:** For DBMSs that default to CONNECTION=UNIQUE, the LIBNAME connection can fail when you use SQL_FUNCTIONS= for that same DBMS to store the external SQL dictionary.

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** ACCESS= LIBNAME option, CONNECTION_GROUP= LIBNAME option, DEFER= LIBNAME option
CAUTION: When you use a GLOBAL or single SHARED connection to access multiple tables, performing a commit or rollback on one table that is being updated might affect all other tables that are open for update. Even if you open a table only for READ, its READ cursor might be resynchronized due to this commit or rollback. If the cursor is resynchronized, there is no guarantee that the new solution table matches the original solution table that was being read.

Syntax

CONNECTION=UNIQUE | SHAREDREAD | SHARED | GLOBALREAD | GLOBAL

Syntax Description

SHAREDREAD

specifies that all READ operations that access DBMS tables in a single libref share a single connection if the conditions for sharing a connection are met. For more information, see “Conditions for a Shared DBMS Connection”.

A separate connection is established for every table that is opened for update or output operations.

Tip Where available, this is usually the default value because it offers the best performance and it guarantees data integrity.

UNIQUE

specifies that a separate connection is established every time a DBMS table is accessed by your SAS application.

Tip Use UNIQUE if you want each use of a table to have its own connection.

SHARED

specifies that all operations that access DBMS tables in a single libref share a single connection if the conditions for sharing a connection are met. For more information, see “Conditions for a Shared DBMS Connection”.

Restriction not valid for MySQL

Note The CONNECTION= option influences only connections that you use to open tables with a libref. When you set CONNECTION=SHARED, it has no influence on utility connections or explicit pass-through connections.

Tip Use SHARED to eliminate the deadlock that can occur when you create and load a DBMS table from an existing table that exists in the same database or table space. This happens only in certain output processing situations and is the only recommended for use with CONNECTION=SHARED.

GLOBALREAD

specifies that all READ operations that access DBMS tables in multiple librefs share a single connection if the conditions for sharing a connection are met. For more information, see “Conditions for a Shared DBMS Connection”.

A separate connection is established for each table that is opened for update or output operations.
GLOBAL
specifies that all operations that access DBMS tables in multiple librefs share a single connection if the conditions for sharing a connection are met. For more information, see “Conditions for a Shared DBMS Connection”.

Restriction
not valid for MySQL

Interactions
One connection is shared for all tables that any libref references for which you specify CONNECTION=GLOBAL.

When CONNECTION_GROUP= is specified, the default value for CONNECTION= is GLOBAL.

Tip
When you set CONNECTION=GLOBAL, any pass-through code that you include after the LIBNAME statement can share the connection. For details, see the CONNECT statement example on page 477 for the pass-through facility.

Details

Overview of the CONNECTION= LIBNAME Option
The main reason for using the CONNECTION= LIBNAME option is to control the number of physical connections to your DBMS. When you specify that you want to share DBMS connections, you enable SAS to use one physical connection across multiple DATA steps and procedure calls. In this way, you limit the number of physical connections for your SAS session. Sharing a connection also enables you to share access to temporary tables across DATA steps and procedure calls.

SAS/ACCESS interfaces that support single or multiple simultaneous connections to the DBMS support this option. Not all values are valid for all SAS/ACCESS interfaces.

For most SAS/ACCESS interfaces, there must be a connection, also known as an attach, to the DBMS server before they can access any data. Typically, each DBMS connection has one transaction, or work unit, that is active in the connection. This transaction is affected by any SQL commits or rollbacks that the engine performs within the connection while executing the SAS application.

The CONNECTION= option lets you control the number of connections, and therefore transactions, that your SAS/ACCESS interface executes and supports for each LIBNAME CONNECT statement, SQL pass-through CONNECT statement, or both.

Table 11.1  DBMS-Specific Default Values

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 under UNIX and PC Hosts, DB2 under z/OS, Impala, Informix, MySQL, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>Aster, Netezza, Vertica</td>
<td>UNIQUE</td>
</tr>
<tr>
<td>Greenplum, HAWQ, Microsoft SQL Server, ODBC, Sybase</td>
<td>UNIQUE for a data source that supports only one active open cursor per connection; otherwise, SHAREDREAD</td>
</tr>
</tbody>
</table>
Conditions for a Shared DBMS Connection

To share a connection across librefs, all of the connection options that you specify must be the same. You can specify the connection options in a LIBNAME statement or in the CONNECT statement in an SQL procedure call. When SAS/ACCESS compares connection option values, it does not matter whether you use optional quotation marks across libref declarations.

Here are the conditions that must be met to share a physical connection to your DBMS.

- These connection options must have the same value for each libref declaration:

  ```
  USER=    SERVER=    DATABASE=
  PASSWORD=  ACCOUNT=  SCHEMA=
  ```

  *Note:* If DATABASE= and SCHEMA= are specified in separate libref declarations, those values must be set to the same value.

- These LIBNAME options must have the same value for each libref declaration:

  ```
  DATABASE=    SQL_FUNCTIONS=
  SCHEMA=    READ_ISOLATION_LEVEL=
  CONNECTION=    READ_LOCK_TYPE=
  CONNECTION_GROUP=    READ_MODE_WAIT=
  DBCONINIT=    UPDATE_ISOLATION_LEVEL=
  DBCONTERM=    UPDATE_LOCK_TYPE=
  DLBLIBINIT=    UPDATE_MODE_WAIT=
  DLBLIBTERM=
  ```

  *Note:* If DATABASE= and SCHEMA= are specified in separate LIBNAME statements, those values must be set to the same value.

If any of these conditions are not met, SAS/ACCESS automatically creates additional physical connections to the DBMS.

Examples

**Example 1: Use SHAREDREAD**

In this example, MYDBLIB makes the first connection to the DBMS. This connection is used to print the data from MYDBLIB.TAB. MYDBLIB2 makes the second connection to the DBMS. A third connection is used to update MYDBLIB.TAB. The third connection is closed at the end of the PROC SQL UPDATE statement. The first and second connections are closed with the CLEAR option.

```sql
/* connection 1 */
libname mydblib oracle user=myusr1
  pw=mypwd1 path='mysrv1'
  connection=sharedread;
```
Example 2: Use GLOBALREAD
In this example, the two librefs, MYDBLIB and MYDBLIB2, share the same connection for Read access because CONNECTION=GLOBALREAD and the connection options are identical. The first connection prints the data from MYDBLIB.TAB while a second connection updates MYDBLIB.TAB. The second connection is closed at the end of the step. The first connection is closed with the final LIBNAME statement.

Example 3: Use UNIQUE
In this example, the MYDBLIB libref establishes a connection to obtain database information. Another connection is established in order to print the data from MYDBLIB.TAB. That connection is closed at the end of the PRINT procedure. Another connection is established to update MYDBLIB.TAB. That connection is closed at the end of the PROC SQL. The CLEAR option in the LIBNAME statement at the end of this example then closes the connection that was made during the MYDBLIB libref assignment.

Example 4: Use SHARED
In this SHARED example, DB2DATA.NEW is created in the database TEST. The DB2DATA.OLD table exists in the same database. So the CONNECTION=SHARED
option lets the DB2 engine share the connection for reading the old table and also creating and loading the new table.

libname db2data db2 connection=shared;
data db2data.new (in = 'database test');
  set db2data.old;
run;

If you did not use the CONNECTION= option in this case, you would deadlock in DB2 and receive this error.

ERROR: Error attempting to CREATE a DBMS table.
ERROR: DB2 execute error DSNT408I SQLCODE = -911,
ERROR: THE CURRENT UNIT OF WORK HAS BEEN ROLLED BACK DUE TO DEADLOCK.

Example 5: Use GLOBAL
In this example for DB2 under z/OS, both PROC DATASETS invocations appropriately report “no members in directory.” This happens because SESSION.B, as a temporary table, has no entry in the SYSIBM.SYSTABLES system catalog. However, the DATA _NULL_ step and SELECT * from PROC SQL step both return the expected rows. For DB2 under z/OS, when SCHEMA=SESSION the database first looks for a temporary table before attempting to access any physical schema named SESSION.

libname x db2 connection=global schema=SESSION;
proc datasets lib=x;
quit;

* DBMS-specific code to create a temporary table impervious to commits and populate the table directly in the DBMS from another table.
*/
proc sql;
connect to db2 (connection=global schema=SESSION);
execute ( DECLARE GLOBAL TEMPORARY TABLE SESSION.B LIKE SASDXS.A 
          ON COMMIT PRESERVE ROWS 
) by db2;
execute ( insert into SESSION.B select * from SASDXS.A 
          ) by db2;
quit;

/* Access the temp table through the global libref. */
data _null_; 
set x.b;
put _all_; 
run;

/* Access the temp table through the global connection. */
proc sql;
connect to db2 (connection=global schema=SESSION);
select * from connection to db2 
( select * from SESSION.B ) ;
quit;
proc datasets lib=x;
quit;

In this example, two different librefs share one connection.

libname db2lib db2 connection=global;
libname db2data db2 connection=global;
If you did not use the CONNECTION= option in this last example, you would deadlock in DB2 and receive this error.

```
ERROR:  Error attempting to CREATE a DBMS table.
ERROR:  DB2 execute error DSNT408I SQLCODE = -911,
ERROR:  THE CURRENT UNIT OF WORK HAS BEEN ROLLED
        BACK DUE TO DEADLOCK.
```

### CONNECTION_GROUP= LIBNAME Option

Causes operations on multiple librefs and on multiple SQL pass-through facility CONNECT statements to share a connection to the DBMS.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement, CONNECT statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>CONNECTION= LIBNAME option</td>
</tr>
</tbody>
</table>

**CAUTION:** When you use a GLOBAL or single SHARED connection for multiple table opens, performing a commit or rollback on one table that is being updated also applies to all other tables that are open for update. Even if you open a table only for READ, its READ cursor might be resynchronized because of this commit or rollback. If the cursor is resynchronized, the new solution table might not match the original solution table that was being read.

### Syntax

```
CONNECTION_GROUP=connection-group-name
```

### Syntax Description

- **connection-group-name**
  - name of a connection group.

### Details

This option causes a DBMS connection to be shared by all operations on multiple librefs if all participating librefs that LIBNAME statements create specify these items:

- the same value for the CONNECTION_GROUP= option
- identical DBMS connection options

If you specify **CONNECTION=GLOBAL** or **CONNECTION=GLOBALREAD=**, operations on multiple librefs can share a connection even if you omit **CONNECTION_GROUP=**.
Informix: The CONNECTION_GROUP option enables multiple librefs or multiple SQL pass-through facility CONNECT statements to share a connection to the DBMS. This overcomes the Release 8.2 limitation where users were unable to access scratch tables across step boundaries as a result of new connections being established with every procedure.

Example: Share a Connection with Identical Connection Options

In this example, the MYDBLIB libref shares a connection with MYDBLIB2 by specifying CONNECTION_GROUP=MYGROUP and by specifying identical connection options. The libref MYDBLIB3 makes a second connection to another connection group called ABC. The first connection is used to print the data from and also for updating MYDBLIB.TAB. The third connection is closed at the end of the step. The first connection is closed by the final LIBNAME statement for that connection. Similarly, the second connection is closed by the final LIBNAME statement for that connection.

```sql
/* connection 1 */
libname mydblib oracle user=myusr1
    pw=mypwd1
    connection_group=mygroup;
libname mydblib2 oracle user=myusr1
    pw=mypwd1
    connection_group=mygroup;
/* connection 2 */
libname mydblib3 oracle user=myusr1
    pw=mypwd1
    connection_group=abc;
proc print data=mydblib.tab...
/* connection 1 */
proc sql;
   update mydblib.tab...
/* does not close connection 1*/
libname mydblib clear;
/* closes connection 1 */
libname mydblib2 clear;
/* closes connection 2 */
libname mydblib3 clear;
```

**CURSOR_TYPE= LIBNAME Option**

Specifies the cursor type for read-only and updatable cursors.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement and some DBMS-specific connection options. See the DBMS-specific reference section for details.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>DBMS- and operation-specific</td>
</tr>
<tr>
<td>Data source:</td>
<td>DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, ODBC, OLE DB, Sybase IQ</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>COMMAND_TIMEOUT= LIBNAME option, CURSOR_TYPE= data set option, KEYSET_SIZE= data set option [only Microsoft SQL Server and ODBC]</td>
</tr>
</tbody>
</table>
Syntax

`CURSOR_TYPE=DYNAMIC | FORWARD_ONLY | KEYSET_DRIVEN | STATIC`

Syntax Description

**DYNAMIC**

specifies that the cursor reflects all changes that are made to the rows in a result set as you move the cursor. The data values and the membership of rows in the cursor can change dynamically on each fetch. This is the default for the DB2 under UNIX and PC Hosts, Microsoft SQL Server, and ODBC interfaces.

**FORWARD ONLY [not valid for OLE DB]**

specifies that the cursor functions like a DYNAMIC cursor except that it supports only sequential fetching of rows.

**KEYSET_DRIVEN**

specifies that the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows are reflected as you scroll around the cursor.

**STATIC**

specifies that the cursor builds the complete result set when the cursor is opened. No changes that are made to the rows in the result set after the cursor is opened are reflected in the cursor. Static cursors are read-only.

Details

Not all drivers support all cursor types. An error is returned if the specified cursor type is not supported. The driver is allowed to modify the default without an error. See your database documentation for more information.

When no options have been set yet, here are the initial DBMS-specific defaults:

<table>
<thead>
<tr>
<th>DB2 under UNIX and PC Hosts</th>
<th>Greenplum</th>
<th>HAWQ</th>
<th>Impala</th>
<th>ODBC</th>
<th>Microsoft SQL Server</th>
<th>Sybase IQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYSET_DRIVEN</td>
<td>FORWARD_ONLY</td>
<td>DYNAMIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here are the operation-specific defaults:
Table 11.3  Operation-Specific Defaults for CURSOR_TYPE=

<table>
<thead>
<tr>
<th>Operation</th>
<th>DB2 under UNIX and PC Hosts</th>
<th>Impala, ODBC, Sybase IQ</th>
<th>Microsoft SQL Server</th>
<th>OLE DB</th>
<th>Greenplum, HAWQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert (UPDATE_SQL=NO)</td>
<td>KEYSET_DRIVEN</td>
<td>KEYSET_DRIVEN</td>
<td>DYNAMIC</td>
<td>FORWARD_ONLY</td>
<td>FORWARD_ONLY</td>
</tr>
<tr>
<td>read (such as PROC PRINT)</td>
<td>driver default</td>
<td></td>
<td></td>
<td>driver default (FORWARD_ONLY)</td>
<td>FORWARD_ONLY</td>
</tr>
<tr>
<td>update (UPDATE_SQL=NO)</td>
<td>KEYSET_DRIVEN</td>
<td>KEYSET_DRIVEN</td>
<td>DYNAMIC</td>
<td>FORWARD_ONLY</td>
<td>FORWARD_ONLY</td>
</tr>
<tr>
<td>CONNECTION=GLOBAL</td>
<td>DYNAMIC</td>
<td></td>
<td></td>
<td>DYNAMIC</td>
<td>FORWARD_ONLY</td>
</tr>
<tr>
<td>CONNECTION=SHARED</td>
<td>DYNAMIC</td>
<td></td>
<td></td>
<td>DYNAMIC</td>
<td>FORWARD_ONLY</td>
</tr>
</tbody>
</table>

OLE DB: Here are the OLE DB properties that are applied to an open rowset. For more information, see your OLE DB programmer reference documentation:

Table 11.4  OLE DB Properties for CURSOR_TYPE= Values

<table>
<thead>
<tr>
<th>CURSOR_TYPE=</th>
<th>OLE DB Properties Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORWARD_ONLY or DYNAMIC (see “Details”)</td>
<td>DBPROP_OTHERINSERT=TRUE, DBPROP_OTHERUPDATEDELETE=TRUE</td>
</tr>
<tr>
<td>KEYSET_DRIVEN</td>
<td>DBPROP_OTHERINSERT=FALSE, DBPROP_OTHERUPDATEDELETE=TRUE</td>
</tr>
<tr>
<td>STATIC</td>
<td>DBPROP_OTHERINSERT=FALSE, DBPROP_OTHERUPDATEDELETE=FALSE</td>
</tr>
</tbody>
</table>

DATETIME2= LIBNAME Option

 Specifies the scale for the timestamp literal for Microsoft SQL Server 2008 and the native Microsoft driver.

Valid in: SAS/ACCESS LIBNAME statement

Default: NO

Data source: ODBC

Note: Support for this LIBNAME option was added for SAS 9.4.

See: DATETIME2= data set option
Syntax

\texttt{DATETIME2=} \texttt{YES} \mid \texttt{NO}

**Syntax Description**

\textbf{YES}

specifies a DATETIME precision and scale when creating the timestamp for the WHERE clause.

\textbf{NO}

uses the first occurring DATETIME precision and scale when creating the timestamp for the WHERE.

---

**DB\_LENGTH\_SEMANTICS\_BYTE= LIBNAME Option**

Indicates whether CHAR and VARCHAR2 column lengths are specified in bytes or characters when creating an Oracle table.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** YES
- **Supports:** NLS
- **Data source:** Oracle
- **See:** DBSERVER\_MAX\_BYTES= LIBNAME option

**Syntax**

\texttt{DB\_LENGTH\_SEMANTICS\_BYTE=} \texttt{YES} \mid \texttt{NO}

**Syntax Description**

\textbf{YES}

specifies that CHAR and VARCHAR2 column lengths are specified in bytes when creating an Oracle table. The byte length is derived by multiplying the number of characters in SAS with DBSERVER\_MAX\_BYTES= value.

\textbf{NO}

specifies that CHAR and VARCHAR2 column lengths are specified in characters when creating an Oracle table. The CHAR keyword is also added next to the length value to indicate that this is the character (not byte) length. For fixed-width encoding, the number of characters is derived by dividing the byte length in SAS for the variable by the value in DBCLIENT\_MAX\_BYTES=. For variable-width encoding, the number of characters remains the same as the number of bytes.

**Details**

This option is appropriate only when creating Oracle tables from SAS. It is therefore ignored in other contexts, such as reading or updating tables.

Length values chosen for variable-width encodings might be more than what is actually needed.
DB_OBJECTS= LIBNAME Option

Specifies which database objects to return with PROC DATASETS or in SAS Explorer.

**Valid in:** SAS/ACCESS LIBNAME statement
**Default:** (TABLES VIEWS)
**Restriction:** Because SHOW_SYNONYMS=YES overrides DB_OBJECTS= and is available only for backward compatibility, you should instead use DB_OBJECT=SYNONYMS.
**Data source:** Oracle
**See:** SHOW_SYNONYMS= LIBNAME option

### Syntax

```
DB_OBJECTS= TABLES | VIEWS | SYNONYMS | ALL
```

### Syntax Description

- **TABLES**
  - Returns only database table names.

- **VIEWS**
  - Returns only database view names.

- **SYNONYMS**
  - Returns only database synonym names.

- **ALL**
  - Returns all database object names, which can slow performance. Specify ALL by itself. It always overrides any multiple values that you specify.

### Example: Specify Multiple Objects

```
DB_OBJECTS= (VIEWS SYNONYMS)
```

### DBCLIENT_ENCODING_FIXED= LIBNAME Option

Specifies whether SAS session encoding is a fixed width.

**Valid in:** SAS/ACCESS LIBNAME statement
**Category:** Data Set Control
**Default:** YES or NO, based on SAS encoding
**Restriction:** Applies only when you create Oracle tables from within SAS
**Supports:** NLS
**Data source:** Oracle
**Note:** Support for this LIBNAME option was added in the first maintenance release for SAS 9.4.
See:
ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS= LIBNAME option,
DB_LENGTH_SEMANTICS_BYTE= LIBNAME option, DBCLIENT_MAX_BYTES= LIBNAME option, DBSERVER_ENCODING_FIXED= LIBNAME option,
DBSERVER_MAX_BYTES= LIBNAME option

CAUTION: To avoid truncation issues, use care when setting this option to YES.

Syntax
DBCLIENT_ENCODING_FIXED=YES | NO

Syntax Description

YES
indicates that database table column lengths in characters are a fixed width. Use this setting to adjust byte lengths within SAS for any database column lengths that are specified in bytes. The number of characters is calculated as the length that is specified in SAS, divided by the value in DBCLIENT_MAX_BYTES= . If needed, and if DB_LENGTH_SEMANTICS_BYTE=YES, the character length is then multiplied by the value in DBSERVER_MAX_BYTES= .

NO
indicates that database table column lengths are not a fixed width.

Example: Set SAS Session Encoding

For this example, the SAS session encoding is euc-cn, the locale is Chinese, and the Oracle server encoding is UTF8.

```sas
libname lib1 oracle path=mypath1 user=myusr1 pw=mypwd1
dbserver_max_bytes=3 dbclient_max_bytes=2 dbclient_encoding_fixed=yes;

data lib1.test;
id='中文';
run;

SQL> desc test;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
<td>CHAR(6)</td>
</tr>
</tbody>
</table>

```sql
libname lib2 oracle path=nlsbip08 user=myusr1 pw=mypwd1
dbserver_max_bytes=3 dbclient_max_bytes=2 dbclient_encoding_fixed=no;

SQL> desc test;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
<td>CHAR(12)</td>
</tr>
</tbody>
</table>
**DBCLIENT_MAX_BYTES= LIBNAME Option**

Specifies the maximum number of bytes per single character in the database client encoding, which matches SAS encoding.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Set Control
- **Alias:** DB_CLIENT_MAXBYTES= [Impala]
- **Defaults:** matches the maximum bytes per single character of SAS session encoding (Oracle, PostgreSQL, Vertica)
  - 1 (Teradata)
- **Supports:** NLS
- **Data source:** Impala, Oracle, PostgreSQL, Teradata, Vertica
- **Note:** Support for Teradata and Vertica was added for SAS 9.4. Support for PostgreSQL was added in the first maintenance release for SAS 9.4.
- **See:** ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS= LIBNAME option, ADJUST_NCHAR_COLUMN_LENGTHS= LIBNAME option, DBSERVER_MAX_BYTES= LIBNAME option
- **Example:** DBSERVER_MAX_BYTES= LIBNAME option

### Syntax

**DBCLIENT_MAX_BYTES=** max-client-bytes

### Required Argument

*max-client-bytes*

specifies the multiplying factor to apply for storage of CHAR and NCHAR variables for client encoding.

### Details

Use this option as the multiplying factor to adjust column lengths for CHAR and NCHAR columns for client encoding. In most cases, you do not need to set this option because the default is sufficient.

For NLS, if DBCLIENT_MAX_BYTES=1, only single-byte characters are supported. For multi-byte characters, the recommendation is to set the value greater than 1. For SAS session encoding UTF-8, set DBCLIENT_MAX_BYTES=3. For the ECU-CN encoding, set DBCLIENT_MAX_BYTES=2.

### Examples

**Example 1: Use Default Values for All Options**

This example uses default values for all options.

```
libname x1 &engine &connopt;
proc contents data=x1.char_sem; run;
proc contents data=x1.nchar_sem; run;
```
### Example 2: Specify Different Settings for Various Options (#1)

Here is one example of this.

```sas
libname x5 &engine &connopt ADJUST_NCHAR_COLUMN_LENGTHS=NO
                   ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=NO
                   DBCLIENT_MAX_BYTES=3;
proc contents data=x5.char_sem; run;
proc contents data=x5.nchar_sem; run;
proc contents data=x5.byte_sem; run;
proc contents data=x5.mixed_sem; run;
```

This next example also uses different settings for the various options.

```sas
libname x6 &engine &connopt ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES
                   ADJUST_NCHAR_COLUMN_LENGTHS=YES
                   DBCLIENT_MAX_BYTES=3;
proc contents data=x6.char_sem; run;
proc contents data=x6.nchar_sem; run;
proc contents data=x6.byte_sem; run;
proc contents data=x6.mixed_sem; run;
```

---

**DBCOMMIT= LIBNAME Option**

Causes an automatic COMMIT (permanently writing data to the DBMS) after processing a specified number of rows.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Alias:** CHECKPOINT= [Teradata]
- **Default:** 1000 when a table is created and rows are inserted in a single step (DATA STEP); 0 when rows are inserted, updated, or deleted from an existing table (PROC APPEND or PROC SQL inserts, updates, or deletes)

**Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Informix, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** BULKLOAD= data set option, DBCOMMIT= data set option, ERRLIMIT= data set option, Maximizing Teradata Load Performance, ML_CHECKPOINT= data set option, Using FastLoad

---

**Syntax**

`DBCOMMIT=n`

**Syntax Description**

`n` specifies the number of rows that are processed. This value must be an integer greater than or equal to 0.
Details

DBCOMMIT= affects update, delete, and insert processing. The number of rows that are processed includes rows that are not processed successfully. Usually, when you set DBCOMMIT=0, COMMIT is issued only once: after a procedure or DATA step completes. However, the commit is performed after each statement when you use the SQL procedure.

If you explicitly set this option, SAS/ACCESS fails any update with a WHERE clause.

Note: If you specify both DBCOMMIT= and ERRLIMIT=, and these options collide during processing, then COMMIT is issued first and ROLLBACK is issued second. Because COMMIT is issued (through the DBCOMMIT= option) before ROLLBACK (through the ERRLIMIT= option), DBCOMMIT= overrides ERRLIMIT=.

DB2 under UNIX and PC Hosts: When BULKLOAD=YES, the default is 10000.

Teradata: For more information, see “FastLoad Supported Features and Restrictions”.

DBCOMMIT= and ERRLIMIT= are disabled for MultiLoad to prevent any conflict with the ML_CHECKPOINT= data set option.

---

DBCONINIT= LIBNAME Option

Specifies a user-defined initialization command to execute immediately after every connection to the DBMS that is within the scope of the LIBNAME statement or libref.

Valid in: SAS/ACCESS LIBNAME statement [all], CONNECT statement [Netezza]

Aliases: DBINITCMD [Vertica]
         INITCMD [Vertica]

Default: none

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: DBCONTERM= LIBNAME option

---

Syntax

```
DBCONINIT=’<DBMS-user-command>’
```

Syntax Description

**DBMS-user-command**

any valid command that the SAS/ACCESS engine can execute and that does not return a result set or output parameters.

Details

The initialization command that you select can be a stored procedure or any DBMS SQL statement that might provide additional control over the interaction between your SAS/ACCESS interface and the DBMS.
The command executes immediately after each DBMS connection is successfully established. If the command fails, a disconnection occurs and the libref is not assigned. You must specify the command as a single quoted string.

**Note:** The initialization command might execute more than once because one LIBNAME statement might have multiple connections (for example, one for reading and one for updating).

**Examples**

**Example 1: Apply the SET Statement to Every Connection**

In this example, the DBCONINIT= option causes the DBMS to apply the SET statement to every connection that uses the MYDBLIB libref.

```
libname mydblib db2
dbconinit="SET CURRENT SQLID='myauthid';"
proc sql;
   select * from mydblib.customers;
   insert into mydblib.customers
      values('33129804', 'VA', '22809', 'USA',
            '540/545-1400', 'BRNETT SUPPLIES', 'M. JONES',
            '2199 LAUREL ST', 'ELKTON', '22APR97'd);
   update mydblib.invoices
      set amtbill = amtbill*1.10
      where country = 'USA';
quit;
```

**Example 2: Pass a Stored Procedure**

In this example, a stored procedure is passed to DBCONINIT=.

```
libname mydblib oracle user=myusr1 pass=mypwd1
dbconinit="begin dept_test(1001,25);
end;"
```

The SAS/ACCESS engine retrieves the stored procedure and executes it.

**Example 3: Treat Backslash Characters as Literals**

In this example, specify that a backslash character (\) should be read as a literal character rather than as an escape character. By default, the DBMS variable that controls how the backslash is read is disabled, resulting in a backslash being treated as an escape character. If this is not the desired behavior (such as when specifying a directory path), you can change the behavior.

The command that you specify varies based on your DBMS. For Greenplum, specify the following command for DBCONINIT=:

```
   dbconinit="SET standard_conforming_strings = 'ON'"
```

For Aster, specify this command for DBCONINIT=:

```
   dbconinit="set session enable_backslash_escapes='off'
```

For MySQL, specify this command for DBCONINIT=:

```
   dbconinit="set session sql_mode='no_backslash_escapes'
```
DBCONTERM= LIBNAME Option

Specifies a user-defined termination command to execute before every disconnect from the DBMS that is within the scope of the LIBNAME statement or libref.

Valid in: SAS/ACCESS LIBNAME statement, CONNECT statement

Aliases: DBTERMCMD [Vertica]
TERMCMD [Vertica]

Default: none

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: DBCONINIT= LIBNAME option

Syntax

DBCONTERM=’<DBMS-user-command>’

Syntax Description

DBMS-user-command
any valid command that the SAS/ACCESS engine can execute and that does not return a result set or output parameters.

Details

The termination command that you select can be a stored procedure or any DBMS SQL statement that might provide additional control over the interaction between the SAS/ACCESS engine and the DBMS. The command executes immediately before SAS terminates each connection to the DBMS. If the command fails, SAS provides a warning message, but unassigning the library and disconnecting from the DBMS still occur. You must specify the command as a single quoted string.

Note: The termination command might execute more than once because one LIBNAME statement might have multiple connections (for example, one for reading and one for updating).

Examples

Example 1: Drop a Table Before Disconnecting
In this example, the DBMS drops the Q1_SALES table before SAS disconnects from the DBMS.

libname mydblib db2 user=myusr1 using=mypwd1
data src=invoice dbconterm='drop table q1_sales';
Example 2: Execute a Stored Procedure at Each DBMS Connection
In this example, the stored procedure, SALESTAB_STORED_PROC, is executed each
time SAS connects to the DBMS, and the BONUSES table is dropped when SAS
terminates each connection.

libname mydblib db2 user=myusr1
   using=mypwd1 datasource=sales
dbconinit='exec salestab_stored_proc'
dbconterm='drop table bonuses';

**DBCREATE_TABLE_EXTERNAL= LIBNAME Option**
Specifies the type of table to create and how associated data files are handled.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Aliases:** DBCREATE_EXTERNAL=
  DBCREATE_EXT=
- **Default:** NO
- **Interaction:** You can specify this option, the DBCREATE_TABLE_LOCATION= option, or both.
- **Data source:** Hadoop
- **Tip:** This option determines only the disposition of a file upon delete.
- **See:** DBCREATE_TABLE_EXTERNAL= data set option,
  DBCREATE_TABLE_LOCATION= data set option, DBCREATE_TABLE_OPTS=
  LIBNAME option, DBCREATE_TABLE_OPTS= data set option

**Syntax**

DBCREATE_TABLE_EXTERNAL=YES | NO

**Syntax Description**

YES
creates an *external* table—one that is stored outside of the Hive warehouse.

NO
creates a *managed* table—one that is managed within the Hive warehouse.

**Details**

When a managed table is dropped, its data is also deleted. When an external table is
dropped, its data is preserved. Create an EXTERNAL table if you want to preserve table
data if the table is dropped. SAS issues a DROP TABLE statement when PROC
DELETE references a Hive table and also with the DROP TABLE statement in PROC
SQL.

**Example: Protect Data from DROP TABLE**

In this example, DBCREATE_TABLE_LOCATION= stores the table data outside of the
Hive warehouse. DBCREATE_TABLE_EXTERNAL=YES protects the data from being
deleted if the table is dropped.
LIBNAME db HADOOP SERVER=mysrv1 USER=myusr1 DB=myschema1;
DATA db.mytab (  
   DBCREATE_TABLE_EXTERNAL=YES  
   DBCREATE_TABLE_LOCATION="/mydir/mytab"  
);
SET mydata;
RUN;

DBCREATE_TABLE_OPTS= LIBNAME Option

Specifies DBMS-specific syntax to add to the CREATE TABLE statement.

Valid in: SAS/ACCESS LIBNAME and CONNECT statements

Alias: POST_STMT_OPTS=

Default: none

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

Tips: If you are already using DBTYPE= within an SQL CREATE TABLE statement, you can also use it to include column modifiers. If you want all output tables to be in the default (non-TEXTFILE) format, see the examples in this section.

See: DBCREATE_TABLE_EXTERNAL= LIBNAME option, DBCREATE_TABLE_EXTERNAL= data set option, DBCREATE_TABLE_LOCATION= data set option, DBCREATE_TABLE_OPTS= data set option, DBTYPE= data set option, POST_STMT_OPTS= data set option, POST_TABLE_OPTS= data set option, PRE_STMT_OPTS= data set option, PRE_TABLE_OPTS= data set option

Syntax

DBCREATE_TABLE_OPTS=’DBMS-SQL-clauses’

Required Argument

DBMS-SQL-clauses specifies one or more DBMS-specific clauses that can be appended to the end of an SQL CREATE TABLE statement.

Details

You can use this option to add DBMS-specific clauses to the end of the SQL CREATE TABLE statement. The SAS/ACCESS engine passes the SQL CREATE TABLE statement and its clauses to the DBMS. The DBMS then executes the statement and creates the DBMS table. This option applies only when you are creating a DBMS table by specifying a libref that is associated with DBMS data.

If you need to add an option in a location other than at the end of your CREATE TABLE statement, use one of these data set options: POST_TABLE_OPTS=, PRE_STMT_OPTS=, and PRE_TABLE_OPTS=. For example, for Greenplum, a WITH clause should appear after the table name but before a DISTRIBUTED RANDOMLY
clause in a CREATE TABLE statement. You should therefore specify a WITH clause using the POST_TABLE_OPTS= data set option.

Examples

**Example 1: Create All Hive Tables in ORC Format**
libname x hadoop ... DBCREATE_TABLE_OPTS="stored as ORC";

**Example 2: Create All Hive Tables in RCFILE Format**
libname x hadoop ... DBCREATE_TABLE_OPTS="stored as RCFILE";

**Example 3: Create All Hive Tables in SEQUENCEFILE Format**
libname x hadoop ... DBCREATE_TABLE_OPTS="stored as SEQUENCEFILE";

---

**DBGEN_NAME= LIBNAME Option**

Specifies how SAS automatically renames to valid SAS variable names any DBMS columns that contain characters that SAS does not allow.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Default:** DBMS
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** DBGEN_NAME= data set option, VALIDVARNAME= system option

---

**Syntax**

`DBGEN_NAME=DBMS | SAS`

**Syntax Description**

**DBMS**

specifies that SAS renames DBMS columns to valid SAS variable names. SAS converts to underscores any characters that it does not allow. If it converts a column to a name that already exists, it appends a sequence number at the end of the new name.

**SAS**

specifies that SAS converts DBMS columns that contain characters that SAS does not allow into valid SAS variable names. SAS uses the format _COLn_, where n is the column number, starting with 0. If SAS converts a name to a name that already exists, it appends a sequence number at the end of the new name.

**Details**

SAS retains column names when it reads data from DBMS tables unless a column name contains characters that SAS does not allow, such as $ or @. SAS allows alphanumeric characters and the underscore (_).
This option is intended primarily for National Language Support, notably for the conversion of kanji to English characters. English characters that are converted from kanji are often those that SAS does not allow. Although this option works for the single-byte character set (SBCS) version of SAS, SAS ignores it in the double-byte character set (DBCS) version. So if you have the DBCS version, you must first set VALIDVARNAME=ANY before using your language characters as column variables.

**Example**

If you specify DBGEN_NAME=SAS, SAS renames a DBMS column named `Dept $Amt` to `_COLn`. If you specify DBGEN_NAME=DBMS, SAS renames the `Dept$Amt` column to `Dept_Amt`.

---

**DBINDEX= LIBNAME Option**

Improves performance when processing a join that involves a large DBMS table and a small SAS data set.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaults:</td>
<td>YES (Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, PostgreSQL, Sybase IQ, Vertica)</td>
</tr>
<tr>
<td></td>
<td>NO (Informix, MySQL, OLE DB, Oracle, SAP HANA, Sybase, Teradata)</td>
</tr>
<tr>
<td>Restriction:</td>
<td>Oracle: Use this option only when the object is a TABLE, not a VIEW. Use DBKEY= when you do not know whether the object is a TABLE.</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>DBINDEX= data set option, “Using the DBINDEX=, DBKEY=, and MULTI_DATASRC_OPT= Options”</td>
</tr>
</tbody>
</table>

**Syntax**

`DBINDEX=YES | NO`

**Syntax Description**

**YES**

specifies that SAS uses columns that have defined DBMS indexes in the WHERE clause.

**NO**

specifies that SAS does not use indexes that are defined on DBMS columns.

**Details**

When you process a join that involves a large DBMS table and a relatively small SAS data set, you might be able to use DBINDEX= to improve performance.

**CAUTION:** Improper use of this option can degrade performance.
**DBLIBINIT= LIBNAME Option**

Specifies a user-defined initialization command to execute once within the scope of the LIBNAME statement or libref that established the first connection to the DBMS.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** none
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** CONNECTION= LIBNAME option, CONNECTION_GROUP= LIBNAME option on page 124, DBCONINIT= LIBNAME option on page 133, DBCONTERM= LIBNAME option on page 135, DLBLIBTERM= LIBNAME option, DEFER= LIBNAME option

**Syntax**

`DBLIBINIT='dbms-user-command'`

**Syntax Description**

**DBMS-user-command**

any DBMS command that the SAS/ACCESS engine can execute and that does not return a result set or output parameters.

**Details**

The initialization command that you select can be a script, stored procedure, or any DBMS SQL statement that might provide additional control over the interaction between your SAS/ACCESS interface and the DBMS.

The command executes immediately after the first DBMS connection is successfully established. If the command fails, a disconnection occurs and the libref is not assigned. You must specify the command as a single quoted string unless it is an environment variable.

`DBLIBINIT=` fails if either `CONNECTION=UNIQUE` or `DEFER=YES`, or if both of these LIBNAME options are specified.

When multiple LIBNAME statements share a connection, the initialization command executes only for the first LIBNAME statement, immediately after the DBMS connection is established. (Multiple LIBNAME statements that use `CONNECTION=GLOBALREAD` and identical values for `CONNECTION_GROUP=`, `DBCONINIT=`, `DBCONTERM=`, `DBLIBINIT=`, and `DBLIBTERM=` options and any DBMS connection options can share the same connection to the DBMS.)

**Example: Allow Only One LIBNAME Statement to Connect**

In this example, `CONNECTION=GLOBALREAD` on page 119 is specified in both LIBNAME statements, but the `DBLIBINIT` commands are different. Therefore, the second LIBNAME statement fails to share the same physical connection.
libname mydblib oracle user=myusr1 pass=mypwd1 connection=globalread dblibinit='Test';
libname mydblib2 oracle user=myusr1 pass=mypwd1 connection=globalread dblibinit='NoTest';

### DLBLTERM= LIBNAME Option

Specifies a user-defined termination command to execute once, before the DBMS that is associated with the first connection made by the LIBNAME statement or libref disconnects.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** none
- **Interaction:** DLBLTERM= fails if either CONNEXION=UNIQUE or DEFER=YES, or if both of these LIBNAME options are specified.
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** CONNECTION= LIBNAME option, CONNECTION_GROUP= LIBNAME option, DBCONINIT= LIBNAME option, DBCONTERM= LIBNAME option, DLBLINIT= LIBNAME option, DEFER= LIBNAME option

### Syntax

**DBLIBTERM=**<>'DBMS-user-command'[

### Syntax Description

**DBMS-user-command**

any DBMS command that can be executed by the SAS/ACCESS engine and that does not return a result set or output parameters.

### Details

The termination command that you select can be a script, stored procedure, or any DBMS SQL statement that might provide additional control over the interaction between the SAS/ACCESS engine and the DBMS. The command executes immediately before SAS terminates the last connection to the DBMS. If the command fails, SAS provides a warning message, but unassigning the library and disconnecting from the DBMS still occur. You must specify the command as a single quoted string.

When two LIBNAME statements share the same physical connection, the termination command is executed only once. CONNECTION=GLOBALREAD and identical values for CONNECTION_GROUP=, DBCONINIT=, DBCONTERM=, DLBLINIT=, and DLBLTERM= options and any DBMS connection options can share the same connection to the DBMS.
Example: Allow Only One LIBNAME Statement to Connect

In this example, CONNECTION=GLOBALREAD is specified in both LIBNAME statements, but the DBLIBTERM commands are different. Therefore, the second LIBNAME statement fails to share the same physical connection.

```plaintext
libname mydblib oracle user=myusr1 pass=mypwd1
   connection=globalread dblibterm='Test';
libname mydblib2 oracle user=myusr1 pass=mypwd1
   connection=globalread dblibterm='NoTest';
```

**DBLINK= LIBNAME Option**

For Oracle, this option specifies a link from your local database to database objects on another server. For Sybase, this option specifies a link from the default database to another database on the server to which you are connected.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** none
- **Data source:** Oracle, Sybase
- **See:** DBMASTER= data set option

**Syntax**

```
DBLINK=database-link
```

**Details**

*Oracle:* A link is a database object that you use to identify an object stored in a remote database. It contains stored path information. It might also contain user name and password information for connecting to the remote database. If you specify a link, SAS uses it to access remote objects. If you omit this option, SAS accesses objects in only the local database.

*Syrbase:* This option lets you link to another database within the same server to which you are connected. If you omit this option, SAS can access objects in only your default database.

**DBMAX_TEXT= LIBNAME Option**

Determines the length of any very long DBMS character data type that is read into SAS or written from SAS when using a SAS/ACCESS engine.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Alias:** TEXTSIZE [Vertica]
- **Default:** 1024 [Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica], 32767 [Hadoop]
- **Restriction:** This option applies to appending and updating rows in an existing table. It does not apply when creating a table.
Requirement: You must set the value to 4000 when you are using procedures that work with SAS High-Performance Analytics Server.

Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, Hadoop, HAWQ, Impala, MySQL, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica

Notes: Support for HAWQ was added in the third maintenance release for SAS 9.4. If you set the value of DBMAX_TEXT= so that data in a table is truncated, the data load fails for that table. The number of bytes that are used to store characters might vary and is based on your session encoding.

Tip: DBMAX_TEXT= is usually used with a very long DBMS character data type, such as the Sybase TEXT data type or the Oracle CLOB data type.

See: DBMAX_TEXT= data set option

Syntax

DBMAX_TEXT=integer

Syntax Description

integer

An integer between 1 and 32,767.

Details

Hadoop: This option applies for the STRING data type.

Oracle: For SAS 9 or higher, this option applies for CLOB, BLOB, LONG, LONG RAW, and LOB data types. The behavior of the ACCESS and DBLOAD procedures have not changed since SAS 8. So only LONG and LOB data types are valid if you use this option with those procedures. Also, this option is ignored if the data type is CHAR or VARCHAR2.

DBMSTEMP= LIBNAME Option

Specifies whether SAS creates temporary or permanent tables.

Valid in: SAS/ACCESS LIBNAME statement

Default: NO

Requirement: To specify this option, you must first specify CONNECTION=GLOBAL—except for Microsoft SQL Server, which defaults to UNIQUE.

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: CONNECTION= LIBNAME option, Temporary Table Support for SAS/ACCESS

Syntax

DBMSTEMP=YES | NO
Syntax Description

**YES**
specifies that SAS creates one or more temporary tables.

**NO**
specifies that SAS creates permanent tables.

Details

To significantly improve performance, you must also set `DBCOMMIT=0`. The value for `SCHEMA=` is ignored. You can then access and use the DBMS temporary tables using SAS/ACCESS engine librefs that share the global connection that SAS used to create those tables.

To join a temporary table and a permanent table, you need a libref for each table and these librefs must successfully share a global connection.

*DB2 under z/OS, Oracle, Teradata:* Set `INSERTBUFF=1000` or higher to significantly improve performance.

*ODBC:* This engine supports DB2, MS SQL Server, or Oracle if you are connected to them.

Example: Create and Join a Permanent Table and a Temporary Table

This example shows how to use this option to create a permanent and temporary table and then join them in a query. The temporary table might not exist beyond a single PROC step. However, this might not be true for all DBMSs.

```sas
options sastrace=(,,d,d) nostsuffix sastraceloc=saslog;
LIBNAME permdata DB2 DB=MA40 SCHEMA=SASTDATA connection=global
dbcommit=0 USER=sasuser PASSWORD=xxx;
LIBNAME tempdata DB2 DB=MA40 SCHEMA=SASTDATA connection=global
dbcommit=0 dbmstemp=yes USER=sasuser PASSWORD=xxx;
proc sql;
create table tempdata.ptyacc as
  (select pty.pty_id
   from permdata.pty_rb pty,
    permdata.PTY_ARNG_PROD_RB acc
   where acc.ACC_PD_CTGY_CD = 'LOC'
    and acc.pty_id = pty.pty_id
   group by pty.pty_id having count(*) > 5 );
create table tempdata.ptyacloc as
  (select ptyacc.pty_id,
   acc.ACC_APPSYS_ID,
   acc.ACC_CO_NO,
   acc.ACCNO,
   acc.ACC_SUB_NO,
   acc.ACC_PD_CTGY_CD
   from tempdata.ptyacc ptyacc,
    permdata.PTY_ARNG_PROD_RB acc
   where ptyacc.pty_id = acc.pty_id
    and acc.ACC_PD_CTGY_CD = 'LOC'
);```
create table tempdata.righttab as

( select ptyacloc.pty_id
from permdata.loc_acc loc,
     tempdata.ptyacloc ptyacloc
where
  ptyacloc.ACC_APPSYS_ID  = loc.ACC_APPSYS_ID
and ptyacloc.ACC_CO_NO      = loc.ACC_CO_NO
and ptyacloc.ACCNO          = loc.ACCNO
and ptyacloc.ACC_SUB_NO     = loc.ACC_SUB_NO
and ptyacloc.ACC_PD_CTGY_CD = loc.ACC_PD_CTGY_CD
and loc.ACC_CURR_LINE_AM - loc.ACC_LDGR_BL > 20000
);

select * from tempdata.ptyacc
except
select * from tempdata.righttab;
drop table tempdata.ptyacc;
drop table tempdata.ptyacloc;
drop table tempdata.righttab;
quit;

DBNULLKEYS= LIBNAME Option

Controls the format of the WHERE clause when you use the DBKEY= data set option.

Valid in: SAS/ACCESS LIBNAME statement

Defaults: YES [Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, Sybase IQ, Vertica]
NO [Informix]
none [Greenplum, HAWQ]

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, Sybase IQ, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: DBKEY= data set option, DBNULLKEYS= data set option

Syntax

DBNULLKEYS= YES | NO

Required Arguments

YES
specifies that there might be NULL values in the key columns in a transaction table or a master table.

NO
specifies that there are no NULL values in the key columns for a transaction table or a master table.
Details

If there might be NULL values in the transaction table or the master table for the
columns that you specify in the **DBKEY=** data set option, use **DBNULLKEYS=YES**.
This is the default for most interfaces. When you specify **DBNULLKEYS=YES** and also
a column that is not defined as NOT NULL in **DBKEY=**, SAS generates a WHERE
clause that can find NULL values. For example, if you specify **DBKEY=COLUMN** and
**COLUMN** is not defined as NOT NULL, SAS generates a WHERE clause with this
syntax:

```
WHERE ((COLUMN = ?) OR ((COLUMN IS NULL) AND (? IS NULL)))
```

With this syntax SAS can prepare the statement once and use it for any (NULL or NOT
NULL) value in the column. This syntax can potentially be much less efficient than the
shorter form of the WHERE clause below. When you specify **DBNULLKEYS=NO** or a
column that **DBKEY=** defines as NOT NULL, SAS generates a simple WHERE clause.

If you know that there are no NULL values in transaction or master tables for the
columns that you specify in the **DBKEY=** option, you can use **DBNULLKEYS=NO**.
This is the default for the Informix interface. If you specify **DBNULLKEYS=NO** and
**DBKEY=COLUMN**, SAS generates a shorter form of the WHERE clause, regardless of
whether the column that is specified in **DBKEY=** is defined as NOT NULL.

```
WHERE (COLUMN = ?)
```

---

**DBPROMPT= LIBNAME Option**

Specifies whether SAS displays a window that prompts the user to enter DBMS connection information
before connecting to the DBMS in interactive mode.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Default:** NO
- **Restriction:** The maximum password length for most of the SAS/ACCESS LIBNAME interfaces is
  32 characters.
- **Interaction:** The **DBPROMPT=** option interacts with the **DEFER=LIBNAME** option to determine
  when the prompt window appears. If **DEFER=NO**, the **DBPROMPT=** window appears
  when the LIBNAME statement is executed. If **DEFER=YES**, the **DBPROMPT=** window appears when you first open a table or view. The **DEFER=** option normally
  defaults to NO, but it defaults to YES if **DBPROMPT=**. You can override this
default by explicitly setting **DEFER=NO**.
- **Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Informix,
  Microsoft SQL Server, MySQL, Netezza, ODBC, Oracle, PostgreSQL, SAP HANA,
  Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** **DBPROMPT= data set option**, **DEFER= LIBNAME option**

**Syntax**

```
DBPROMPT=YES | NO
```
Syntax Description

YES
specifies that SAS displays a window that interactively prompts you for the DBMS connection options the first time the libref is used.

NO
specifies that SAS does not display the prompting window.

Details

If you specify DBPROMPT=YES, it is not necessary to provide connection options with the LIBNAME statement. If you use the LIBNAME statement to specify connection options and DBPROMPT=YES, connection option values are displayed in the window. The value of the password appears as a series of asterisks. You can override all of these values interactively.

The DBPROMPT window usually opens only once for each time that the LIBNAME statement is specified. It might open multiple times if DEFER=YES and the connection fails when SAS tries to open a table. In such cases, the DBPROMPT window appears until a successful connection occurs or you click Cancel.

Oracle: You can enter 30 characters for the USERNAME and PASSWORD and up to 70 characters for the PATH, depending on your platform.

Teradata: You can enter up to 30 characters for the USERNAME and PASSWORD.

Examples

Example 1: Preventing a Prompt Window from Opening
In this example, the DBPROMPT window does not open when the LIBNAME statement is submitted because DEFER=YES. The DBPROMPT window appears when the PRINT procedure is processed, a connection is made, and the table is opened.

```
libname mydblib oracle dbprompt=yes defer=yes;
proc print data=mydblib.staff;
run;
```

Example 2: Allow a Prompt Window to Open Only Once
In this example, the DBPROMPT window appears while the LIBNAME statement is processing. The DBPROMPT window does not open in subsequent statements because the DBPROMPT window appears only once per LIBNAME statement.

```
libname mydblib oracle dbprompt=yes defer=no;
```

Example 3: Allow Values to Appear in a Prompt Window
In this example, values provided in the LIBNAME statement are pulled into the DBPROMPT window. The values myusr1 and mysrv1 appear in the DBPROMPT window, and the user can edit and confirm them. The password value appears in the DBPROMPT window as a series of asterisks, so the user can also edit it.

```
libname mydblib oracle
  user=myusr1 pw=mypwd1
  path='mysrv1' dbprompt=yes defer=no;
```
DBSASLABEL= LIBNAME Option

Specifies the column labels an engine uses.

Valid in: SAS/ACCESS LIBNAME statement
Default: COMPAT

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

Tip: You can use this option to override the default behavior. It is useful for when PROC SQL uses column labels as headers instead of column aliases.

See: DBSASLABEL= data set option

Syntax

```
DBSASLABEL=COMPAT | NONE
```

Syntax Description

COMPAT
specifies that the labels returned should be compatible with what the application normally receives—meaning that engines exhibit their normal behavior.

NONE
specifies that the engine does not return a column label. The engine returns blanks for the column labels.

Details

By default, the SAS/ACCESS interface for your DBMS generates column labels from the column names instead of from the real column labels.

Example: Return Blank Labels for Aliases in Headings

This example shows how to use DBSASLABEL= as a LIBNAME option to return blank column labels so that PROC SQL can use the column aliases as the column headings.

```
libname x oracle user=myusr1 pw=mypwd1;
proc sql;
  select deptno as Department ID, loc as Location
  from mylib.dept(dbsaslabel=none);
```

Without DBSASLABEL=NONE, aliases are ignored, and DEPTNO and LOC are used as column headings in the result set.

DBSERVER_ENCODING_FIXED= LIBNAME Option

Specifies whether Oracle database encoding is a fixed width.

Valid in: SAS/ACCESS LIBNAME statement
**Category:** Data Set Control

**Default:** YES or NO, based on Oracle server encoding

**Restriction:** Applies only when you read Oracle tables in SAS

**Interaction:** To avoid truncation issues, use care when setting this option to YES.

**Supports:** NLS

**Data source:** Oracle

**Note:** Support for this option was added in the first maintenance release for SAS 9.4.

**See:**
- ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS= LIBNAME option,
- DB_LENGTH_SEMANTICS_BYTE= LIBNAME option,
- DBCLIENT_ENCODING_FIXED= LIBNAME option,
- DBCLIENT_MAX_BYTES= LIBNAME option,
- DBSERVER_MAX_BYTES= LIBNAME option

**CAUTION:** To avoid truncation issues, use care when setting this option to YES.

---

**Syntax**

**DBSERVER_ENCODING_FIXED**=YES | NO

**Syntax Description**

**YES**

- indicates that database table column lengths in characters are a fixed width. Use this setting to adjust byte lengths within SAS for any database column lengths that are specified in bytes. The number of characters is calculated by dividing the byte length by the value in **DBSERVER_MAX_BYTES**.

**NO**

- indicates that database table column lengths are not a fixed width.

---

**Example: Set Oracle Database Encoding**

For this example, the SAS session encoding is euc-cn, the locale is Chinese, and the Oracle server encoding is UTF8.

```sas
/* Read */
/* Prepare in Oracle a table with char(6), named test. */
/* See from SQLPLUS: */
SQL> desc test;
<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
<td>CHAR(6)</td>
</tr>
</tbody>
</table>
```

```sas
libname lib1 oracle path=nlsbip08 user=myusr1 pw=mypwd1
dbserver_max_bytes=3 dbclient_max_bytes=2
dbserver_encoding_fixed = yes;
```

```sas
proc contents data=lib1.test;run;
```
libname lib2 oracle path=nlsbip08 user=myusr1 pw=mypwd1
dbserver_max_bytes=3 dbclient_max_bytes=2
dbserver_encoding_fixed=no;

proc contents data=lib2.test;run;

Result

First Proc contents
Alphabetic List of Variables and Attributes
#  Variable  Type  Len  Format  Informat  Label
1   ID       Char  4    $4.    $4.     ID

Second Proc contents
Alphabetic List of Variables and Attributes
#  Variable  Type  Len  Format  Informat  Label
1   ID       Char  12   $12.   $12.   ID

### DBSERVER_MAXBYTES= LIBNAME Option

Specifies the maximum number of bytes per single character in the database server encoding.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Set Control
- **Alias:** DB_MAX_BYTES= [Impala]
- **Default:** usually 1
- **Supports:** NLS
- **Data source:** DB2 under UNIX and PC Hosts, Impala, Oracle, Sybase, Vertica
- **See:**
  - ADJUST_BYTE SEMANTIC_COLUMN_LENGTHS= LIBNAME option,
  - ADJUST_NCHAR_COLUMN_LENGTHS= LIBNAME option,
  - DB_LENGTH_SEMANTICS_BYTE= LIBNAME option, DBCLIENT_MAX_BYTES= LIBNAME option
- **Example:** DBCLIENT_MAX_BYTES= LIBNAME option

#### Syntax

**DBSERVER_MAXBYTES=max-server-bytes**

#### Details

Use this option to derive (adjust the value of) the number of characters from the client column lengths that byte semantics initially creates. Although the default is usually 1, you can use this option to set it to another value if this information is available from the Oracle server.

**Sybase:** You can use this option to specify different byte encoding between the SAS client and the Sybase server. For example, if the client uses double-byte encoding and the server uses multibyte encoding, specify DBSERVER_MAXBYTES=3. In this case, the SAS/ACCESS engine evaluates this option only if you specify a value that is greater than 2. Otherwise, it indicates that both client and server use the same encoding scheme.
Examples

**Example 1: Adjust Specific Column Lengths**

Only the lengths that you specify with DBSERVER_MAX_BYTES= affect column lengths that byte semantics created initially.

```sas
libname x4 &engine &connopt DBSERVER_MAX_BYTES=4
  DBCLIENT_MAX_BYTES=1 ADJUST_NCHAR_COLUMN_LENGTHS=no;
proc contents data=x4.char_sem; run;
proc contents data=x4.nchar_sem; run;
proc contents data=x4.byte_sem; run;
proc contents data=x4.mixed_sem; run;
```

**Example 2: Specify Different Settings for Various Options (#1)**

```sas
libname x5 &engine &connopt ADJUST_NCHAR_COLUMN_LENGTHS=NO
  ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=NO DBCLIENT_MAX_BYTES=3;
proc contents data=x5.char_sem; run;
proc contents data=x5.nchar_sem; run;
proc contents data=x5.byte_sem; run;
proc contents data=x5.mixed_sem; run;
```

**Example 3: Specify Different Settings for Various Options (#2)**

```sas
libname x6 &engine &connopt ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=YES
  ADJUST_NCHAR_COLUMN_LENGTHS=YES DBCLIENT_MAX_BYTES=3;
proc contents data=x6.char_sem; run;
proc contents data=x6.nchar_sem; run;
proc contents data=x6.byte_sem; run;
proc contents data=x6.mixed_sem; run;
```

**DBSLICEPARM= LIBNAME Option**

Controls the scope of DBMS threaded Reads and the number of threads.

Valid in: SAS/ACCESS LIBNAME statement (also available as a SAS configuration option, SAS invocation option, global SAS option, or data set option)

Defaults: None [Vertica]

- THREADED_APPS, none [Greenplum, HAWQ]
- THREADED_APPS=2 [DB2 under z/OS, Oracle, Teradata]
- THREADED_APPS=2or 3 [DB2 under UNIX and PC Hosts, Informix, Microsoft SQL Server, ODBC, SAP HANA, Sybase, Sybase IQ]

Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, ODBC, Oracle, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Notes: Support for Vertica was added for SAS 9.4. Support for SAP HANA was added in the first maintenance release for SAS 9.4. Support for Greenplum was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: DBSLICE= data set option, DBSLICEPARM= data set option, DBSLICEPARM= system option, SLEEP= LIBNAME option, SLEEP= data set option, TENACITY= LIBNAME option, TENACITY= data set option
Syntax

\[
\text{DBSLICEPARM}= \text{NONE} | \text{THREADING\_APPS} | \text{ALL} \\
\text{DBSLICEPARM}=( \text{NONE} | \text{THREADING\_APPS} | \text{ALL} < \text{max-threads}> ) \\
\text{DBSLICEPARM}=( \text{NONE} | \text{THREADING\_APPS} | \text{ALL}<, \text{max-threads}> )
\]

Syntax Description

NONE

disables DBMS threaded Read. SAS reads tables on a single DBMS connection, as it
did with SAS 8 and earlier.

THREADING\_APPS

makes fully threaded SAS procedures (threaded applications) eligible for threaded
Reads.

ALL

makes all read-only librefs eligible for threaded Reads. This includes SAS threaded
applications, as well as the SAS DATA step and numerous SAS procedures.

max-threads

a positive integer value that specifies the maximum number of connections per table
read. The second parameter of the option determines the number of threads to read
the table in parallel. The number of partitions on the table determine the number of
connections made to the Oracle server for retrieving rows from the table. A partition
or portion of the data is read on each connection. The combined rows across all
partitions are the same regardless of the number of connections. That is, changes to
the number of connections do not change the result set. Increasing the number of
connections instead redistributes the same result set across more connections.

If the database table is not partitioned, SAS creates max-threads number of
connections with WHERE MOD()... predicates and the same number of threads.

There are diminishing returns when increasing the number of connections. With each
additional connection, more burden is placed on the DBMS, and a smaller
percentage of time saved on the SAS step. See the DBMS-specific reference section
for details about partitioned reads before using this parameter.

Details

You can use DBSLICEPARM= in numerous locations. The usual rules of option
precedence apply: A table option has the highest precedence, then a LIBNAME option,
and so on. SAS configuration file option has the lowest precedence because
DBSLICEPARM= in any of the other locations overrides that configuration setting.

DBSLICEPARM=ALL and DBSLICEPARM=THREADING\_APPS make SAS programs
eligible for threaded Reads. To see whether threaded Reads are actually generated, turn
on SAS tracing and run a program, as shown in this example.

options sastrace=",,t" sastraceloc=saslog nostsuffix;
proc print data=lib.dbtable(dbsliceparm=(ALL));
  where dbcol>1000;
run;

If you want to directly control the threading behavior, use the DBSLICE= data set
option.
Greenplum, HAWQ: There is no default value for the maximum number of connections per table read. This value depends on the number of partitions in a table and on arguments that are used with the MOD function in a WHERE clause. For more information, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

DB2 under UNIX and PC Hosts, Informix, Microsoft SQL Server, ODBC, Sybase, Sybase IQ: The default thread number depends on whether an application passes in the number of threads (CPUCOUNT=) and whether the data type of the column that was selected for data partitioning is binary.

Examples

Example 1: Disable Threaded Read for All SAS Users
Here is how to use DBSLICEPARM= in a SAS configuration file entry in Windows to turn off threaded Reads for all SAS users.

-dbsliceparm NONE

Example 2: Enable Threaded Reads for Read-Only References
Here is how you can use DBSLICEPARM= as a z/OS invocation option to turn on threaded Reads for read-only references to DBMS tables throughout a SAS job.

sas o(dbsliceparm=ALL)

Example 3: Increase Maximum Threads (as a SAS Global Option)
In this example, you can use DBSLICEPARM= as a SAS global option to increase maximum threads to three for SAS threaded applications. This OPTIONS statement is typically one of the first statements in your SAS code.

options dbsliceparm=(threaded_apps,3);

Example 4: Enable Threaded Reads for References Using a Particular Libref
You can use DBSLICEPARM= as a LIBNAME option to turn on threaded Reads for read-only table references that use this particular libref, as shown in this example.

libname dblib oracle user=myusr1 password=mypwd1 dbsliceparm=ALL;

Example 5: Enable Threaded Reads as a Table-Level Option
Here is how to use DBSLICEPARM= as a table-level option to turn on threaded Reads for this particular table, requesting up to four connections.

proc reg SIMPLE;
   data=dblib.customers (dbsliceparm=(all,4));
   var age weight;
   where years_active>1;
run;

DEFER= LIBNAME Option

Specifies when the connection to the DBMS occurs.

Valid in: SAS/ACCESS LIBNAME statement, CONNECT statement
The default value of NO is overridden if DBPROMPT=YES.
The DEFER= option is ignored when CONNECTION=UNIQUE because a connection is performed every time a table is opened.

Microsoft SQL Server, Netezza, ODBC: When you set DEFER=YES, you must also set the PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options to the values that you want. Normally, SAS queries the data source to determine the correct defaults for these options during LIBNAME assignment, but setting DEFER=YES postpones the connection. Because these values must be set at the time of LIBNAME assignment, you must assign them explicitly when you set DEFER=YES.

Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: CONNECTION= LIBNAME option, DBPROMPT= LIBNAME option

**Syntax**

```
DEFER=YES | NO
```

**Syntax Description**

**NO** specifies that the connection to the DBMS occurs when the libref is assigned by a LIBNAME statement.

**YES** specifies that the connection to the DBMS occurs when a table in the DBMS is opened.

**DEGREE= LIBNAME Option**

Determines whether DB2 uses parallelism.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>ANY</td>
</tr>
<tr>
<td>Data source</td>
<td>DB2 under z/OS</td>
</tr>
<tr>
<td>See</td>
<td>DEGREE= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

```
DEGREE=ANY | 1
```
**Syntax Description**

**ANY**

enables DB2 to use parallelism, and issues the SET CURRENT DEGREE = 'xxx' for all DB2 threads that use that libref.

**1**

explicitly disables the use of parallelism.

**Details**

When DEGREE=ANY, DB2 has the option of using parallelism, when it is appropriate. Setting DEGREE=1 prevents DB2 from performing parallel operations. Instead, DB2 is restricted to performing one task that, although this is perhaps slower, it uses less system resources.

**DELETE_MULT_ROWS= LIBNAME Option**

Indicates whether to let SAS delete multiple rows from a data source, such as a DBMS table.

**Valid in:** SAS/ACCESS LIBNAME statement

**Default:** NO

**Data source:** Aster, Greenplum, HAWQ, Microsoft SQL Server, Netezza, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase IQ, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** UPDATE_MULT_ROWS= LIBNAME option

**Syntax**

DELETE_MULT_ROWS=YES | NO

**Syntax Description**

**YES**

specifies that SAS/ACCESS processing continues if multiple rows are deleted. This might produce unexpected results.

**NO**

specifies that SAS/ACCESS processing does not continue if multiple rows are deleted.

**Details**

Some providers do not handle this DBMS SQL statement well and therefore delete more than the current row:

DELETE...WHERE CURRENT OF CURSOR

**DIMENSION= LIBNAME Option**

Specifies whether the database creates dimension tables or fact tables.
Valid in: SAS/ACCESS LIBNAME statement
Default: NO
Data source: Aster
See: DIMENSION= data set option, PARTITION_KEY= LIBNAME option, PARTITION_KEY= data set option

Syntax
DIMENSION=YES | NO

Syntax Description
YES
  specifies that the database creates dimension tables.
NO
  specifies that the database creates fact tables.

DIRECT_EXE= LIBNAME Option
Lets an SQL delete statement be passed directly to a DBMS with pass-through.
Valid in: SAS/ACCESS LIBNAME statement
Default: none
Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
Tip: Performance improves significantly by using DIRECT_EXE=. This is because the SQL delete statement is passed directly to the DBMS instead of having SAS read the entire result set and delete one row at a time.
See: DBIDIRECTEXEC= system option

Syntax
DIRECT_EXE=DELETE

Syntax Description
DELETE
  specifies that an SQL delete statement is passed directly to the DBMS for processing.

Example: Empty a Table from a Database
libname x oracle user=myusr1 password=mypwd1
  path=oraclev8 schema=testschema
direct_exe=delete; /* Create an Oracle table of 5 rows. */
data x.dbi_dft;
do col1=1 to 5;
output;
end;
run;

options sastrace=",,,d" sastraceloc=saslog nostsuffix;
proc sql;
delete * from x.dbi_dft;
quit;

By turning trace on, you should see something similar to this:

**Output 11.1 SAS Log Output**

```
ORACLE_9: Executed:
delete from dbi_dft
```

---

**DIRECT_SQL= LIBNAME Option**

Specifies whether generated SQL is passed to the DBMS for processing.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** YES
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** SQL_FUNCTIONS= LIBNAME option

### Syntax

- **DIRECT_SQL=** YES | NO | NONE
- **DIRECT_SQL=** NOGENSQL | NOWHERE | NOFUNCTIONS | NOMULTOUTJOINS

### Syntax Description

**YES**

specifies that generated SQL from PROC SQL is passed directly to the DBMS for processing.

**NO**

specifies that generated SQL from PROC SQL is not passed to the DBMS for processing. This is the same as specifying the value NOGENSQL.

**NONE**

specifies that generated SQL is not passed to the DBMS for processing. This includes SQL that is generated from PROC SQL, SAS functions that can be converted into DBMS functions, joins, and WHERE clauses.

**NOGENSQL**

prevents PROC SQL from generating SQL to be passed to the DBMS for processing.
NOWHERE prevents WHERE clauses from being passed to the DBMS for processing. This includes SAS WHERE clauses and PROC SQL generated or PROC SQL specified WHERE clauses.

NOFUNCTIONS prevents SQL statements from being passed to the DBMS for processing when they contain functions.

NOMULTOUTJOINS specifies that PROC SQL does not attempt to pass any multiple outer joins to the DBMS for processing. Other join statements might be passed down however, including portions of a multiple outer join.

Details

By default, processing is passed to the DBMS whenever possible, because the database might be able to process the functionality more efficiently than SAS does. In some instances, however, you might not want the DBMS to process the SQL. For example, the presence of null values in the DBMS data might cause different results depending on whether the processing takes place in SAS or in the DBMS. If you do not want the DBMS to handle the SQL, use DIRECT_SQL= to force SAS to handle some or all SQL processing.

If you specify DIRECT_SQL=NOGENSQL, PROC SQL does not generate DBMS SQL. This means that SAS functions, joins, and DISTINCT processing that occur within PROC SQL are not passed to the DBMS for processing. (SAS functions outside PROC SQL can still be passed to the DBMS.) However, if PROC SQL contains a WHERE clause, the WHERE clause is passed to the DBMS, if possible. Unless you specify DIRECT_SQL=NOWHERE, SAS attempts to pass all WHERE clauses to the DBMS.

If you specify more than one value for this option, separate the values with spaces and enclose the list of values in parentheses. For example, you could specify DIRECT_SQL=(NOFUNCTIONS NOWHERE).

DIRECT_SQL= overrides the SQL_FUNCTIONS=LIBNAME option. If you specify SQL_FUNCTIONS=ALL and DIRECT_SQL=NONE, no functions are passed.

Examples

Example 1: Prevent a DBMS from Processing a Join

This example prevents the DBMS from processing a join between two tables by setting DIRECT_SQL=NOGENSQL. SAS processes the join instead.

```sas
proc sql;
create view work.v as
  select tab1.deptno, dname from
    mydblib.table1 tab1,
    mydblib.table2 tab2
  where tab1.deptno=tab2.deptno
  using libname mydblib oracle user=myusr1
  password=mypwd1 path=mysrv1 direct_sql=nogensql;
```

Example 2: Prevent a DBMS from Processing a SAS Function

```sas
libname mydblib oracle user=myusr1 password=mypwd1 direct_sql=nofunctions;
proc print data=mydblib.tab1;
  where lastname=soundex ('Paul');
```
DRIVER_VENDOR= LIBNAME Option

specifies the name of the ODBC driver vendor.

Valid in: SAS/ACCESS LIBNAME statement, CONNECT statement
Default: CLOUDERA
Interaction: Use the SAS_IMPALA_DRIVER_VENDOR environment variable for the entire SAS session. When both are set, the LIBNAME option has precedence over the environment variable.

Data source: Impala

Note: Support for this LIBNAME option was added in the third maintenance release for SAS 9.4.

Examples: Set the LIBNAME option:

libname imp impala user=myuser1 password=mypwd1 server=myimpalaserver
   schema=myschema driver_vendor=cloudera;

Set the environment variable on PC hosts:

SAS_IMPALA_DRIVER_VENDOR CLOUDERA

Export the environment variable on UNIX hosts:

export SAS_IMPALA_DRIVER_VENDOR=CLOUDERA

Set the environment variable at SAS invocation for PC and UNIX:

sas -dms -set SASIMPALA_DRIVER_VENDOR CLOUDERA

Syntax

DRIVER_VENDOR=<CLOUDERA> | <DATADIRECT> | <MAPR> | <PROGRESS>

Optional Arguments

CLOUDERA
   specifies Cloudera as the ODBC driver vendor.
DATADIRECT
   specifies DataDirect as the ODBC driver vendor.
MAPR
   specifies MapR as the ODBC driver vendor.
PROGRESS
   specifies Progress as the ODBC driver vendor.

ENABLE_BULK= LIBNAME Option

Lets the connection process bulk copy when loading data into a Sybase table.

Valid in: SAS/ACCESS LIBNAME statement
Default: YES
**Syntax**

ENABLE_BULK=YES | NO

**Syntax Description**

NO

- disables bulk copy ability for the libref.

YES

- lets the connection perform bulk copy of SAS data into Sybase.

**Details**

Bulk copy groups rows so that they are inserted as a unit into the Sybase table. Using bulk copy can improve performance.

If you use both the, ENABLE_BULK= LIBNAME option and the BULKLOAD= data set option, values for both options must be the same or an error is returned. However, because ENABLE_BULK=YES is the default value, you do not need to specify ENABLE_BULK= to use the BULKLOAD= data set option.

---

**ERRLIMIT= LIBNAME Option**

Specifies the number of errors that are allowed while using the Fastload utility before SAS stops loading data to Teradata.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>DATA and PROC steps (wherever Fastload is used)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1 million</td>
</tr>
</tbody>
</table>

**Data source:** Teradata

**See:** ERRLIMIT= data set option, DBCOMMIT= LIBNAME option, DBCOMMIT= data set option, ML_CHECKPOINT= data set option

---

**Syntax**

ERRLIMIT=integer

**Syntax Description**

integer

- specifies a positive integer that represents the number of errors after which SAS stops loading data.

**Details**

SAS stops loading data when it reaches the specified number of errors and Fastload pauses. When Fastload pauses, you cannot use the table that is being loaded. Restart
capability for Fastload is not yet supported, so you must manually delete the error tables before SAS can reload the table.

Example
In this example, SAS stops processing and pauses Fastload when it encounters the tenth error.

```sas
libname mydblib teradata user=terau user=XXXXXX ERRLIMIT=10;
data mydblib.trfload(bulkload=yes dbtype=(i='int check (i > 11)'));
do
   i=1 to 50000;output;
end;
run;
```

ESCAPE_BACKSLASH= LIBNAME Option

Specifies whether backslashes in literals are preserved during data copy from a SAS data set to a table.

| Valid in: | SAS/ACCESS LIBNAME statement |
| Default: | NO |
| Data source: | MySQL |

**Syntax**

`ESCAPE_BACKSLASH=YES | NO`

**Syntax Description**

**YES**

- specifies that an additional backslash is inserted in every literal value that already contains a backslash.

**NO**

- specifies that backslashes that exist in literal values are not preserved. An error results.

**Details**

MySQL uses the backslash as an escape character. When data that is copied from a SAS data set to a MySQL table contains backslashes in literal values, the MySQL interface can preserve these if `ESCAPE_BACKSLASH=YES`.

FASTEXPORT= LIBNAME Option

Specifies whether the SAS/ACCESS engine uses the TPT API to read data.

| Valid in: | SAS/ACCESS LIBNAME statement |
| Default: | NO |
| Data source: | Teradata |
Syntax

FASTEXPORT=YES | NO

Syntax Description

YES
specifies that the SAS/ACCESS engine should use the Teradata Parallel Transporter (TPT) API to read data from a Teradata table.

NO
specifies that the SAS/ACCESS engine does not use the TPT API to read data from a Teradata table.

Details

By using the TPT API, you can read data from a Teradata table without working directly with the stand-alone Teradata FastExport utility. When FASTEXPORT=YES, SAS uses the TPT API export driver for bulk reads. If SAS cannot use the TPT API—due to an error or because it is not installed on the system—it still tries to read the data. However, it does not produce an error. To check whether SAS used the TPT API to read data, look for this message in the SAS log:

```
NOTE:  Teradata connection:  TPT FastExport has read n row(s).
```

When you specify a query band on this option, you must set the DBSLICEPARM=LIBNAME option. The query band is passed as a SESSION query band to the FastExport utility.

To see whether threaded Reads are actually generated, turn on SAS tracing by setting OPTIONS SASTRACE=",,,d" in your program.

Example

In this example, the TPT API reads SAS data from a Teradata table. SAS still tries to read data even if it cannot use the TPT API.

```
Libname tera Teradata user=myusr1 pw=mypwd1 FASTEXPORT=YES;
/* Create data */
Data tera.testdata;
Do i=1 to 100;
  Output;
End;
Run;
/* Read using FastExport TPT. This note appears in the SAS log if SAS uses TPT.
NOTE:  Teradata connection:  TPT FastExport has read n row(s). */
```
Data work.testdata;
Set tera.testdata;
Run;

**FASTLOAD= LIBNAME Option**

Quickly loads large amounts of data into an empty Teradata table.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Alias:** BULKLOAD=
- **Default:** NO
- **Data source:** Teradata
- **See:** BULKLOAD= LIBNAME option, BULKLOAD= data set option, DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option, DBSLICEPARM= system option, LOGDB= LIBNAME option, FASTEXPORT= LIBNAME option FASTLOAD= data set option Maximizing Teradata Load Performance, MULTILOAD= data set option, QUERY_BAND= LIBNAME option, QUERY_BAND= data set option, SLEEP= LIBNAME option, SLEEP= data set option, TENACITY= LIBNAME option, TENACITY= data set option

**Syntax**

```
FASTLOAD YES | NO
```

**Syntax Description**

**YES**

specifies that the SAS/ACCESS engine use the FastLoad protocol to load the data

**NO**

specifies that the SAS/ACCESS engine does not use the FastLoad protocol

**Details**

You can specify FASTLOAD= using a data set or LIBNAME option. Use care with the FASTLOAD= LIBNAME option (rather than the data set option). Any time you insert data from SAS into Teradata tables, the FastLoad protocol is used. This uses Teradata utility slots and might also cause other load jobs to fail.

Setting the FASTLOAD= LIBNAME option on a SAS library hides it from users. In a BI environment, it can be difficult to tell whether the option is set. If you are setting up SAS libraries that are used only by ETL jobs, then it might be acceptable to use the LIBNAME option.

**Note:** A best practice recommendation is to use `FASTLOAD=` as a data set option unless you have a compelling reason to use it as a LIBNAME option.

```sas
libname mytera TERADATA server=teraserv user=bob pw=bob1;
/* Create and load a table using a Data step. SAS numeric is */
/* forced to be an INTEGER. */
data mytera.table0 (FASTLOAD=YES DBTYPE= (x= INTEGER));
   do x = 1 to 1000000;
      output;
   end;
run;
```
PROC SQL;
   CONNECT TO TERADATA (USER=bob PW=bob1 SERVER=teraserv5500);
   EXECUTE (DROP TABLE loadThisTable) BY TERADATA;
   EXECUTE (COMMIT) BY TERADATA;
   EXECUTE (CREATE MULTISET TABLE loadThisTable (a INTEGER, b CHAR(10))
             PRIMARY INDEX (a)) BY TERADATA;
   EXECUTE (COMMIT) BY TERADATA;
QUIT;
DATA work.loadData;
   FORMAT b $10.;
   a = 1;
   output;
   b = 'One';
   output;
   a = 2;
   output;
   b = 'Two';
   output;
   a = 3;
   output;
   b = 'Three';
   output;
RUN;
libname mytera teradata server=teraserv user=bob pw=bob1;
PROC APPEND BASE=mytera.loadThisTable (FASTLOAD=YES BL_LOG=BOB_APPEND_ERR)
   DATA=work.loadData;
RUN;

**FETCH_IDENTITY**= LIBNAME Option

Returns the value of the last inserted identity value.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>NO</td>
</tr>
<tr>
<td>Data source:</td>
<td>DB2 under UNIX and PC Hosts</td>
</tr>
<tr>
<td>See:</td>
<td>FETCH_IDENTITY= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

FETCH_IDENTITY=YES | NO

**Syntax Description**

**YES**

returns the value of the last inserted identity value.

**NO**

disables this option.
Details

You can use this option instead of issuing a separate SELECT statement after an INSERT statement. If FETCH.IDENTITY=YES and the INSERT that is executed is a single-row INSERT, the engine calls the DB/2 identity_val_local() function and places the results into the SYSDB2_LAST_IDENTITY macro variable. Because the DB2 engine default is multirow inserts, you must set INSERTBUFF=1 to force a single-row INSERT.

HDFS_PRINCIPAL= LIBNAME Option

Specifies the Kerberos principal for HDFS.

Valid in: SAS/ACCESS LIBNAME statement

Alias: HDFS_KERBEROS_PRINCIPAL=

Default: none

Restriction: The HDFS_PRINCIPAL= LIBNAME option applies only when BULKLOAD=YES and only when you configure HDFS to allow Kerberos authentication.

Data source: Impala

Note: Support for this LIBNAME option was added in the second maintenance release for SAS 9.4.

See: BULKLOAD= LIBNAME option, IMPALA_PRINCIPAL= LIBNAME option

Syntax

HDFS_PRINCIPAL=’principal’

Required Argument

principal

specifies the server’s Kerberos service principal name (SPN). Surround the principal value with single or double quotation marks. For example, you might specify a principal value that is similar to the following code:

hdfs_principal='hdfs/hdfs_host.example.com@TEST.EXAMPLE.COM'

IGNORE_READ_ONLY_COLUMNS= LIBNAME Option

Specifies whether to ignore or include columns where data types are read-only when generating an SQL statement for inserts or updates.

Valid in: SAS/ACCESS LIBNAME statement

Default: NO

Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase IQ, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: IGNORE_READ_ONLY_COLUMNS= data set option
Syntax

\texttt{IGNORE\_READ\_ONLY\_COLUMNS=\texttt{YES | NO}}

\textbf{Syntax Description}

\textbf{YES}

specifies that the \textsc{sas/access} engine ignores columns where data types are read-only when you are generating insert and update SQL statements.

\textbf{NO}

specifies that the \textsc{sas/access} engine does not ignore columns where data types are read-only when you are generating insert and update SQL statements.

\textbf{Details}

Several databases include data types that can be read-only, such as the data type of the Microsoft SQL Server timestamp. Several databases also have properties that allow certain data types to be read-only, such as the Microsoft SQL Server identity property.

When \texttt{IGNORE\_READ\_ONLY\_COLUMNS=NO} and a DBMS table contains a column that is read-only, an error is returned indicating that the data could not be modified for that column.

\textbf{Example}

For this example, a database that contains the table \texttt{Products} is created with two columns: \texttt{ID} and \texttt{PRODUCT\_NAME}. The \texttt{ID} column is defined by a read-only data type and \texttt{PRODUCT\_NAME} is a character column.

\begin{verbatim}
CREATE TABLE products (id int IDENTITY PRIMARY KEY, product_name varchar(40))
\end{verbatim}

Assume you have a SAS data set that contains the name of your products, and you would like to insert the data into the \texttt{Products} table.

\begin{verbatim}
data work.products;
  id=1;
  product_name='screwdriver';
  output;
  id=2;
  product_name='hammer';
  output;
  id=3;
  product_name='saw';
  output;
  id=4;
  product_name='shovel';
  output;
run;
\end{verbatim}

With \texttt{IGNORE\_READ\_ONLY\_COLUMNS=NO} (the default), an error is returned by the database because in this example the \texttt{ID} column cannot be updated. However, if you set the option to \texttt{YES} and execute a \texttt{PROC APPEND}, the append succeeds, and the SQL statement that is generated does not contain the \texttt{ID} column.

\begin{verbatim}
libname x odbc uid=myusr1 pwd=myusr1 dsn=lupinss
  ignore_read_only_columns=yes;
options sastrace=',,\'d' sastraceloc=saslog nostsuffix;
proc append base=x.PRODUCTS data=work.products;
\end{verbatim}
**IMPALA_PRINCIPAL= LIBNAME Option**

Specifies the Kerberos principal for the Impala server.

**Valid in:** SAS/ACCESS LIBNAME statement  
**Default:** none

**Restrictions:**  
The IMPALA_PRINCIPAL= LIBNAME option applies only when you connect to the Impala server in a LIBNAME statement.  
The IMPALA_PRINCIPAL= LIBNAME option applies only when you configure Impala to allow Kerberos authentication.

**Data source:** Impala

**Note:** Support for this LIBNAME option was added in the second maintenance release for SAS 9.4.

**See:** HDFS_PRINCIPAL= LIBNAME option, "LIBNAME Statement Specifics for Hadoop"

**Syntax**

`IMPALA_PRINCIPAL='principal'`

**Required Argument**

`principal`

specifies the server’s Kerberos service principal name (SPN). Surround the principal value with single or double quotation marks. For example, you might specify a principal value that is similar to the following code:

```
impala_principal='impala/impala_host.example.com@TEST.EXAMPLE.COM'
```

---

**IN= LIBNAME Option**

Lets you specify the database and table space in which you want to create a new table.

**Valid in:** SAS/ACCESS LIBNAME statement  
**Alias:** TABLESPACE=  
**Default:** none

**Data source:** DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** IN= data set option

**Syntax**

```
IN='database-name.tablespace-name' | 'DATABASE database-name'
```
Syntax Description

database-name.tablespace-name
  specifies the names of the database and table space, which are separated by a period. Enclose the entire specification in single quotation marks.

DATABASE database-name
  specifies only the database name. Specify the word DATABASE, a space, and the database name. Enclose the entire specification in single quotation marks.

Details

The IN= option is relevant only when you are creating a new table. If you omit this option, the default is to create the table in the default database, implicitly creating a simple table space.

INSERT_SQL= LIBNAME Option

Determines the method to use to insert rows into a data source.

Valid in: SAS/ACCESS LIBNAME statement
Default: DBMS-specific
Data source: Greenplum, HAWQ, Microsoft SQL Server, ODBC, OLE DB, Vertica
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: INSERT_SQL= data set option, INSERTBUFF= LIBNAME option, INSERTBUFF= data set option

Syntax

INSERT_SQL=YES | NO

Syntax Description

YES
  specifies that SAS/ACCESS uses the SQL insert method for the data source to insert new rows into a table.

NO
  specifies that SAS/ACCESS uses an alternate, DBMS-specific method to insert new rows into a table.

Details

Flat file databases such as dBASE, FoxPro, and text files generally have improved insert performance when INSERT_SQL=NO. Other databases might have inferior insert performance or might fail with this setting. You should therefore experiment to determine the optimal setting to meet your needs.

Greenplum, HAWQ, ODBC: The default is YES, except for Microsoft Access, where the default is NO. When INSERT_SQL=NO, the SQLSetPos (SQL_ADD) function inserts rows in groups that are the size of the INSERTBUFF= option value. The SQLSetPos (SQL_ADD) function does not work unless your driver supports it.

Microsoft SQL Server: The Microsoft SQL Server default is YES. When INSERT_SQL=NO, the SQLSetPos (SQL_ADD) function inserts rows in groups that
are the size of the INSERTBUFF= option value. The SQLSetPos (SQL_ADD) function
does not work unless your driver supports it.

Netezza, Vertica: The default is YES.

OLE DB: By default, the OLE DB interface tries to use the most efficient row-insertion
method for each data source. You can use the INSERT_SQL option to override the
default in the event that it is not optimal for your situation. Used when this option is set
to NO, the alternate OLE DB method uses the OLE DB IRowsetChange interface.

INSERTBUFF= LIBNAME Option

Specifies the number of rows in a single DBMS insert.

Valid in: SAS/ACCESS LIBNAME statement

Default: DBMS-specific

Restrictions: SAS application messages that indicate the success or failure of an Insert operation
represent information for only a single insert, even when multiple inserts are
performed. Therefore, when you assign a value that is greater than INSERTBUFF=1,
these messages might be incorrect.

When you insert rows with the VIEWTABLE window or the FSVIEW or FSEDIT
procedure, use INSERTBUFF=1 to prevent the DBMS interface from trying to insert
multiple rows. These features do not support inserting more than one row at a time.

Additional driver-specific restrictions might apply.

Interactions: SAS allows the maximum number of rows that the DBMS allows.

If you set the DBCOMMIT= option with a value that is less than the value of
INSERTBUFF=, then DBCOMMIT= overrides INSERTBUFF=.

The optimal value for this option varies with factors such as network type and
available memory.

To use this option, you must set INSERT_SQL=YES, (DB2 under UNIX and PC
Hosts, Greenplum, Microsoft SQL Server)

If one row in the insert buffer fails, all rows in the insert buffer fail. (DB2 under UNIX
and PC Hosts)

Values greater than 0 activate the INSERTBUFF= option, and the engine calculates
how many rows it can insert at one time, based on row size. If one row in the insert
buffer fails, all rows in the insert buffer might fail, depending on your storage type.
(MySQL)

Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL
Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

Tip: You might need to experiment with different values to determine the best value for
your site.

See: INSERTBUFF= data set option, DBCOMMIT= LIBNAME option, DBCOMMIT= data
set option, INSERT_SQL= LIBNAME option, INSERT_SQL= data set option,
READBUFF= LIBNAME option, READBUFF= data set option

Syntax

\texttt{INSERTBUFF=} \texttt{positive-integer}
Syntax Description

*positive-integer*

    specifies the number of rows to insert.

Details

### Table 11.5  DBMS-Specific Default Values

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Netezza, PostgreSQL, SAP HANA, Sybase IQ, Vertica</td>
<td>The default is automatically calculated based on row length.</td>
</tr>
<tr>
<td>MySQL</td>
<td>The default is 0.</td>
</tr>
<tr>
<td>Microsoft SQL Server, ODBC, OLE DB</td>
<td>The default is 1.</td>
</tr>
<tr>
<td>Oracle</td>
<td>When REREAD_EXPOSURE=YES, the (forced) default value is 1. Otherwise, the default is 10.</td>
</tr>
</tbody>
</table>

**INTERFACE= LIBNAME Option**

Specifies the name and location of the interfaces file that is searched when you connect to the Sybase server.

Valid in: SAS/ACCESS LIBNAME statement  
Default: none  
Data source: Sybase

Syntax

**INTERFACE=<>filename<>**

Details

The interfaces file contains names and access information for the available servers on the network. If you omit a filename, the default action for your operating system occurs. INTERFACE= is not used in some operating environments. Contact your database administrator to see whether this statement applies to your computing environment.

**KEYSET_SIZE= LIBNAME Option**

Specifies the number of rows that are keyset-driven.

Valid in: SAS/ACCESS LIBNAME statement and some DBMS-specific connection options. See the DBMS-specific reference section for details.
Syntax

KEYSET_SIZE=number-of-rows

Syntax Description

number-of-rows

an integer with a value between 0 and the number of rows in the cursor.

Details

If KEYSET_SIZE=0, the entire cursor is keyset-driven. If you specify a value greater than 0 for KEYSET_SIZE=, that value indicates the number of rows within the cursor that functions as a keyset-driven cursor. When you scroll beyond the bounds that KEYSET_SIZE= specifies, the cursor becomes dynamic and new rows might be included in the cursor. This becomes the new keyset, and the cursor functions as a keyset-driven cursor again. Whenever you specify a value between 1 and the number of rows in the cursor, the cursor is considered to be a mixed cursor: Part of it functions as a keyset-driven cursor and part functions as a dynamic cursor.

LOCATION= LIBNAME Option

Allows further qualification of exactly where a table resides.

Valid in: SAS/ACCESS LIBNAME statement

Alias: LOC=

Default: none

Requirement: If you specify LOCATION=, you must also specify the AUTHID=LIBNAME option.

Data source: DB2 under z/OS

See: AUTHID= LIBNAME option, LOCATION= data set option, REMOTE_DBTYPE= LIBNAME option [to access a database server on Linux, UNIX, or Windows]

Syntax

LOCATION=location

Details

The location name maps to the location in the SYSIBM.LOCATION catalog in the communication database.

In SAS/ACCESS Interface to DB2 under z/OS, the location is converted to the first level of a three-level table name: location.authid.table. The DB2 Distributed Data Facility
(DDF) makes the connection implicitly to the remote DB2 subsystem when DB2
receives a three-level name in an SQL statement.

If you omit this option, SAS accesses the data from the local DB2 database unless you
have specified a value for the SERVER= option. This option is not validated until you
access a DB2 table.

**LOCKTABLE= LIBNAME Option**

Places exclusive or shared locks on tables.

| Valid in: | SAS/ACCESS LIBNAME statement |
| Default: | no locking |
| Data source: | Informix |

See: [LOCKTABLE= data set option](#)

**Syntax**

```
LOCKTABLE=EXCLUSIVE | SHARE
```

**Syntax Description**

**EXCLUSIVE**

specifies that other users are prevented from accessing each table that you open in
the libref.

**SHARE**

specifies that other users or processes can read data from the tables, but they cannot
update the data.

**Details**

You can lock tables only if you are the owner or have been granted the necessary
privilege.

**LOCKTIME= LIBNAME Option**

Specifies the number of seconds to wait until rows are available for locking.

| Valid in: | SAS/ACCESS LIBNAME statement |
| Default: | none |
| Requirement: | You must specify LOCKWAIT=YES for LOCKTIME= to have an effect. |
| Data source: | Informix |

See: [LOCKWAIT= LIBNAME option](#)

**Syntax**

```
LOCKTIME=positive-integer
```
Details

If you omit the LOCKTIME= option and use LOCKWAIT=YES, SAS suspends your process indefinitely until a lock can be obtained.

LOCKWAIT= LIBNAME Option

Specifies whether to wait indefinitely until rows are available for locking.

- Valid in: SAS/ACCESS LIBNAME statement
- Default: DBMS-specific
- Data source: Informix, Oracle
- See: LOCKTIME= LIBNAME option

Syntax

LOCKWAIT=YES | NO

Syntax Description

- YES
  - specifies that SAS waits until rows are available for locking.

- NO
  - specifies that SAS does not wait and returns an error to indicate that the lock is not available.

LOGDB= LIBNAME Option

Redirects to an alternate database-specific table that FastExport creates or MultiLoad uses.

- Valid in: DATA and PROC steps, wherever you use FastExport or MultiLoad
- Default: default Teradata database for the libref
- Data source: Teradata
- Tip: You can also use LOGDB= with TPT options.
- See: FASTEXPORT= LIBNAME option, MULTILOAD= data set option, TPT= LIBNAME option, TPT= data set option, “Using MultiLoad”

Syntax

LOGDB=database-name

database-name
- specifies the name of the Teradata database.
Details

Teradata Fast Export utility: The FastExport restart capability is not yet supported. When you use this option with FastExport, FastExport creates restart log tables in an alternate database. You must have the necessary permissions to create tables in the specified database, and FastExport creates only restart tables in that database.

Teradata MultiLoad utility: To specify this option, you must first specify MULTILOAD=YES. When you use this option with the Teradata MultiLoad utility, MultiLoad redirects the restart table, the work table, and the required error tables to an alternate database.

Examples

Example 1: Create Restart Log Tables
In this example, PROC PRINT calls the Teradata FastExport utility, if it is installed. FastExport creates restart log tables in the ALTDB database.

libname mydblib teradata user=myusr1 pw=mypwd1 logdb=altdb;
proc print data=mydblib.mytable(dbsliceparm=all);
run;

Example 2: Create Restart, Work, and Error Tables
In this next example, MultiLoad creates the restart table, work table, and error tables in the alternate database that LOGDB= specifies.

    /* Create work tables in altdb2 database, where I have create & drop privileges. */
    libname x teradata user=myusr1 pw=xxxxx logdb=altdb2;
data x.testload(multiload=YES);
do i=1 to 100;
   output;
end;
run;

Example 3: Create the Work Table in a Different Database
Using MultiLoad with the TPT API, this example provides a different name for the work table and redirects the table to the AUDATA00_work database.

libname tera teradata user=myusr1 pw=mypwd1 logdb=audata00_work;
data tera.testdata(MULTILOAD=YES TPT_WORK_TABLE=work);
i=1;output; i=2;output;
run;

LOGIN_TIMEOUT= LIBNAME Option
Specifies the default login time-out for connecting to and accessing data sources in a library.

Valid in: SAS/ACCESS LIBNAME statement, CONNECT statement
Defaults: 30 [Hadoop]
0 [Aster, Impala, Netezza, ODBC, SAP HANA, Sybase IQ]
Data source: Aster, Hadoop, Impala, Netezza, ODBC, SAP HANA, Sybase IQ
Syntax

LOGIN TIMEOUT=numeric-value

Syntax Description

numeric-value

specifies a positive integer for the number of seconds to wait for the connection. A value of 0 indicates to wait indefinitely.

MAX_CONNECTS= LIBNAME Option

Specifies the maximum number of simultaneous connections that Sybase allows.

Valid in: SAS/ACCESS LIBNAME statement
Default: 25
Data source: Sybase

Syntax

MAX_CONNECTS=numeric-value

Details

If you omit MAX_CONNECTS=, the default for the maximum number of connections is 25. Increasing the number of connections has a direct impact on memory.

MODE= LIBNAME Option

Specifies whether the connection to Teradata uses the ANSI or Teradata mode.

Valid in: SAS/ACCESS LIBNAME statement
Default: ANSI
Data source: Teradata
See: SQL Pass-Through Facility Specifics for Teradata

Syntax

MODE=TERADATA | ANSI

Syntax Description

TERADATA

specifies that SAS/ACCESS opens Teradata connections in Teradata mode.

ANSI

specifies that SAS/ACCESS opens Teradata connections in ANSI mode.
Details

This option allows Teradata connections to open in the specified mode. Connections that open with MODE=TERADATA use Teradata mode rules for all SQL requests that are passed to the Teradata DBMS. This impacts transaction behavior and can cause case insensitivity when processing data.

During data insertion, not only is each inserted row committed implicitly, but rollback is not possible when the error limit is reached if you also specify ERRLIMIT=. Any update or deletion that involves a cursor does not work.

ANSI mode is recommended for all features that SAS/ACCESS supports, and Teradata mode is recommended only for reading data from Teradata.

Example

This example does not work because it requires the use of a cursor.

```sas
libname x teradata user=myusr1 pw=XXXX mode=teradata;
/* Fails with "ERROR:  Cursor processing is not allowed in Teradata mode." */
proc sql;
update x.test
set i=2;
quit;
```

This example works because the DBIDIRECTEXEC= system option sends the delete SQL directly to the database without using a cursor.

```sas
libname B teradata user=myusr1 pw=XXXX mode=Teradata;
options dbidirectexec;
proc sql;
delete from b.test where i=2;
quit;
```

MULTI_DATASRC_OPT= LIBNAME Option

Used in place of DBKEY to improve performance when processing a join between two data sources.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>DBMASTER= data set option</td>
</tr>
</tbody>
</table>

Syntax

MULTI_DATASRC_OPT=NONE | IN_CLAUSE
**Syntax Description**

**NONE**

turns off option functionality.

**IN_CLAUSE**

specifies use of an IN clause that contains values that were read from a smaller table. The clause is used to retrieve matching values in a larger table based on a key column that was designated in an equijoin.

**Details**

When processing a join between a SAS data set and a DBMS table, the SAS data set should be smaller than the DBMS table for optimal performance. However, if the SAS data set is larger than the DBMS table, the SAS data set is still used in the IN clause.

When SAS processes a join between two DBMS tables, SELECT COUNT (*) is issued to determine which table is smaller and if it qualifies for an IN clause. You can use the DBMASTER= data set option to prevent the SELECT COUNT (*) from being issued.

The IN clause currently has a limit of 4,500 unique values.

Setting **DBKEY=** automatically overrides **MULTI_DATASRC_OPT=**.

**DIRECT_SQL=** can impact this option as well. If **DIRECT_SQL=NONE** or NOWHERE, the IN clause cannot be built and passed to the DBMS, regardless of the value of **MULTI_DATASRC_OPT=**. These settings for **DIRECT_SQL=** prevent a WHERE clause from being passed.

**Oracle:** Oracle can handle an IN clause of only 1,000 values. It therefore divides larger IN clauses into multiple smaller IN clauses. The results are combined into a single result set. For example, if an IN clause contained 4,000 values, Oracle produces 4 IN clauses that contain 1,000 values each. A single result is produced, as if all 4,000 values were processed as a whole.

**OLE DB:** OLE DB restricts the number of values allowed in an IN clause to 255.

**Examples**

**Example 1: Build and Pass an IN Clause for a Join**

This example builds and passes an IN clause from the SAS table to the DBMS table, retrieving only the necessary data to process the join.

```sql
proc sql;
create view work.v as
select tab2.deptno, tab2.dname from
work.sastable tab1, dblib.table2 tab2
where tab12.deptno = tab2.deptno
using libname dblib oracle user=myusr1 password=mypwd1
   multi_datasrc_opt=in_clause;
quit;
```

**Example 2: Prevent Build and Pass of an IN Clause for a Join**

This example prevents the building and passing of the IN clause to the DBMS. It requires all rows from the DBMS table to be brought into SAS to process the join.

```sql
libname dblib oracle user=myusr1 password=mypwd1 multi_datasrc_opt=none;
proc sql;
   select tab2.deptno, tab2.dname from
```
MULTISTMT= LIBNAME Option

Specifies whether insert statements are sent to Teradata one at a time or in a group.

Valid in: SAS/ACCESS LIBNAME statement
Default: NO
Restriction: You currently cannot use MULTISTMT= with ERRLIMIT=.
Data source: Teradata
See: MULTISTMT= data set option

Syntax
MULTISTMT=YES | NO

Syntax Description
YES
tries to send as many inserts to Teradata that can fit in a 64K buffer. If multistatement inserts are not possible, processing reverts to single-row inserts.

NO
send inserts to Teradata one row at a time.

Details
When you request multistatement inserts, SAS first determines how many insert statements it can send to Teradata. Several factors determine the actual number of statements that SAS can send—for example, how many:

• SQL insert statements can fit in a 64K buffer.
• data rows can fit in the 64K data buffer.
• inserts the Teradata server chooses to accept.

When you need to insert large volumes of data, you can significantly improve performance by using MULTISTMT= instead of inserting only single-row.

If you also specify DBCOMMIT=, SAS uses the smaller of these: the DBCOMMIT= value and the number of insert statements that can fit in a buffer as the number of insert statements to send together at one time.

OR_BINARY_DOUBLE= LIBNAME Option

Specifies the default data type to use for numeric table columns.

Valid in: SAS/ACCESS LIBNAME statement
Default: YES
OR_ENABLE_INTERRUPT= LIBNAME Option

Allows interruption of any long-running SQL processes on the DBMS server.

Valid in: SAS/ACCESS LIBNAME statement
Default: NO
Data source: Oracle

Syntax

OR_ENABLE_INTERRUPT= YES | NO

Syntax Description

YES
allows interruption of long-running SQL processes on the DBMS server.

NO
Disables interruption of long-running SQL processes on the DBMS server.

Details

Beginning in SAS 9.1 of the SAS/ACCESS Interface to DB2, a new SAS environment variable, SAS_ODBC_ENABLE_INTERRUPT, was added to aid in interrupting long-running queries submitted through the SAS interface. You can use this option to interrupt these statements:

• any SELECT SQL statement that was submitted by using the SELECT * FROM CONNECTION as a pass-through statement
any statement other than the SELECT SQL statement that you submitted by using the EXECUTE statement as a pass-through statement

**OR_UPD_NOWHERE= LIBNAME Option**

Specifies whether SAS uses an extra WHERE clause when updating rows with no locking.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Alias:** ORACLE_73_OR_ABOVE=
- **Default:** YES

**Requirement:** Due to the published Oracle bug 440366, an update on a row sometimes fails even if the row has not changed. Oracle offers this solution: When you create a table, increase the number of INITRANS to at least 3 for the table.

**Data source:** Oracle

**See:** "Locking in the Oracle Interface", OR_UPD_NOWHERE= data set option, UPDATE_LOCK_TYPE= LIBNAME option

### Syntax

**OR_UPD_NOWHERE=**YES | NO

### Syntax Description

**YES**

specifies that SAS does not use an additional WHERE clause to determine whether each row has changed since it was read. Instead, SAS uses the SERIALIZABLE isolation level (available with Oracle7.3 and above) for update locking. If a row changes after the serializable transaction starts, the update on that row fails.

**NO**

specifies that SAS uses an additional WHERE clause to determine whether each row has changed since it was read. If a row has changed since being read, the update fails.

### Details

Use this option when you are updating rows without locking (UPDATE_LOCK_TYPE=NOLOCK).

By default (OR_UPD_NOWHERE=YES), updates are performed in serializable transactions. It lets you avoid extra WHERE-clause processing and potential WHERE-clause floating-point precision problems.

**PACKETSIZE= LIBNAME Option**

Allows specification of the packet size for Sybase to use.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** current server setting
- **Data source:** Sybase
Syntax

PACKETSIZE=numeric-value

Syntax Description

numeric-value
  any multiple of 512, up to the limit of the maximum network packet size setting on your server.

Details

If you omit PACKETSIZE=, the default is the current server setting. You can query the default network packet value in the ISQL utility by using the Sybase sp_configure command.

PARMDEFAULT= LIBNAME Option

Specifies whether the SAP HANA engine uses the defaults for variables and parameters that are specified in the metadata in SAP HANA.

Valid in: SAS/ACCESS LIBNAME statement
Alias: USE_PARAMETER_DEFAULT
Data source: SAP HANA

Syntax

PARMDEFAULT=YES | NO

Syntax Description

YES
  specifies to look up metadata for variables and parameters. This enables you to apply defaults for variables in a WHERE clause, and defaults for input parameters using the PLACEHOLDER syntax.

NO
  specifies to not look up metadata for variables and parameters.

Details

The default for the PARMDEFAULT= option is YES if the PARMSTRING=LIBNAME= or data set option is specified. The default is NO if no PARMSTRING= option is specified.

Applying the defaults requires additional queries to the metadata. It can be switched off to avoid making unnecessary queries.

PARMSTRING= LIBNAME Option

Specifies a quoted string of variable name and value pairs separated by a comma, or a placeholder string.
Valid in: SAS/ACCESS LIBNAME statement
Alias: PARAMETERS
Data source: SAP HANA

Syntax

```parmstring = "<"variable-name1=variable-value1,variable-name2=variable-value2,..."> < "PLACEHOLDER" = ("variable-name1","variable-value1"), "PLACEHOLDER" = ("variable-name2","variable-value2"),...">```

Syntax Description

"variable-name1=variable-value1"

specifies the variable name and value pair. More than one pair can be specified, and must be separated by a comma.

"PLACEHOLDER = ("variable-name1","variable-value1")"

specifies the variable name and value pair as a placeholder.

Note: You can also combine a name and value pair and a placeholder:

```parmstring = "variable-name1=variable-value1", < 'PLACEHOLDER' = ('$$parm_product$$','Tablet')"```

Details

When you specify the variable name and value pairs, the SAS/ACCESS engine locates the variable input parameter in the SAP HANA metadata. The value is applied either as a WHERE clause for variables, or it generates and applies a PLACEHOLDER= string for passing the input parameters to the view execution. If the variable input parameter is not found in metadata, it is appended as part of a PLACEHOLDER= string to the generated SQL string.

If the user specifies a placeholder string, the string is passed directly to the SAP HANA query to be processed in SAP HANA.

Here are some syntax examples:

```parmstring = "parm_price=30"
parmstring = "'PLACEHOLDER' = ("$$parm_product$$", 'Tablet')"
parmstring = "PLACEHOLDER. $$parm_category$$:'Notebooks'"
```

When a PARMSTRING= LIBNAME option and a PARMSTRING= data set option are both specified, the PLACEHOLDER= string becomes a combination of both of the parameters and is passed to SAP HANA. An input parameter can occur only once as a placeholder in the SQL statement. If a parameter appears in the LIBNAME option and the data set option, the SAS/ACCESS engine tries to resolve this by passing only a fragment from the data set option.

Comparisons

The PARMSTRING= LIBNAME option is applied to all column tables and column views in the library. It is not applied to row store objects.

The PARMSTRING= data set option is applied to the table or view that is specified.
PARTITION_KEY= LIBNAME Option

Specifies the column name to use as the partition key for creating fact tables.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Requirements: To create a data set in Aster without error, either set DIMENSION= YES for the LIBNAME or data set option or specify a partition key in the PARTITION_KEY= LIBNAME or data set option. You must enclose the column name in quotation marks.

Data source: Aster
See: DIMENSION= LIBNAME option, DIMENSION= data set option, PARTITION_KEY= data set option

Syntax
PARTITION_KEY='column-name'

Details
Aster uses dimension and fact tables.

Example: Create a Dimension Table
This first example shows how you can use the SAS data set, SASFLT. flightschedule, to create an Aster dimension table, flightschedule by using the DIMENSION= DATA step option.

LIBNAME sasflt 'SAS-library';
LIBNAME net_air ASTER user=myusr1 pwd=mypwd1 server=air2 database=flights;
data net_air.flightschedule(dimension=yes);
set sasflt. flightschedule;
run;

You can create the same Aster dimension table by setting DIMENSION=YES in the LIBNAME statement.

LIBNAME sasflt 'SAS-library';
LIBNAME net_air ASTER user=myusr1 pwd=mypwd1 server=air2 database=flights dimension=yes;
data net_air.flightschedule;
set sasflt. flightschedule;
run;

If you do not set DIMENSION=YES by using either the LIBNAME or data set option, the Aster engine tries to create an Aster fact table. To do this, however, you must set the PARTITION_KEY= LIBNAME or data set option, as shown in this example.

LIBNAME sasflt 'SAS-library';
LIBNAME net_air ASTER user=myusr1 pwd=mypwd1 server=air2 database=flights;
data net_air.flightschedule(dbtype=(flightnumber=integer) partition_key='flightnumber');
set sasflt. flightschedule;
run;

You can create the same Aster fact table by using the PARTITION_KEY= LIBNAME option.

LIBNAME sasflt 'SAS-library';
LIBNAME net_air ASTER user=myusr1 pwd=mypwd1 server=air2 database=flights partition_key='flightnumber';
data net_air.flightschedule(dbtype=('flightnumber'=integer));
   set sasflt.flightsschedule;
run;

The above examples use the DBTYPE= data set option so that the data type of the partition-key column meets the limitations of the Aster partition-key column.

### POST_STMT_OPTS= LIBNAME Option

Allows additional database-specific options to be placed after the CREATE TABLE statement in generated SQL code.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>DBCREATE_TABLE_OPTS=</td>
</tr>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for this option was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>DBCREATE_TABLE_OPTS= LIBNAME option, DBIDIRECTEXEC system option, POST_STMT_OPTS= data set option</td>
</tr>
</tbody>
</table>

### Syntax

**POST_STMT_OPTS=’DBMS-SQL-option(s)’**

### Required Argument

**DBMS-SQL-option(s)**

- specifies database-specific options to be placed after the CREATE TABLE statement. Enclose the options that you specify within single or double quotation marks.

### Details

You can use POST_STMT_OPTS= to add DBMS-specific clauses to the end of the SQL CREATE TABLE statement. The SAS/ACCESS engine passes the SQL CREATE TABLE statement and its clauses to the DBMS, which executes the statement and creates the DBMS table. POST_STMT_OPTS= applies only when you are creating a DBMS table by specifying a libref that is associated with DBMS data.
**PRESERVE_COL_NAMES= LIBNAME Option**

Preserves spaces, special characters, and case sensitivity in DBMS column names when you create DBMS tables.

**Valid in:** SAS/ACCESS LIBNAME statement (when you create DBMS tables)

**Alias:** PRESERVE_NAMES= (see “Details”)

**Defaults:**
- YES (MySQL, ODBC to Microsoft SQL Server)
- NO (Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Teradata, Vertica)

**Restrictions:** This option applies only when you use SAS/ACCESS to create a new DBMS table. PRESERVE_COL_NAMES= does not apply to the SQL pass-through facility.

**Interaction:** If you use the DS2 or FedSQL language, quoting and casing of names is different. For more information, see the identifiers topic in SAS DS2 Language Reference or SAS FedSQL Language Reference.

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** PRESERVE_COL_NAMES= data set option, SAS Names and Support for DBMS Names, VALIDVARNAME= system option

---

**Syntax**

```
PRESERVE_COL_NAMES=YES | NO
```

**Syntax Description**

**NO**

specifies that column names that are used to create DBMS tables are derived from SAS variable names (VALIDVARNAME= system option) by using the SAS variable name normalization rules. However, the database applies its DBMS-specific normalization rules to the SAS variable names when creating the DBMS column names.

The use of N-literals to create column names that use database keywords or special symbols other than the underscore character might be invalid when DBMS normalization rules are applied. To include nonstandard SAS symbols or database keywords, specify PRESERVE_COL_NAMES=YES.

NO is the default for most DBMS interfaces.

**YES**

specifies that column names that are used in table creation are passed to the DBMS with special characters and the exact, case-sensitive spelling of the name is preserved.
Details

When you create a table, you assign the column names by using one of these methods.

- To control the case of the DBMS column names, specify variables using the case that you want and set `PRESERVE_COL_NAMES=YES`. If you use special symbols or blanks, you must set `VALIDVARNAME=TO ANY` and use N-literals. For more information, see the SAS/ACCESS naming topic in the DBMS-specific reference section for your interface in this document and also SAS Data Set Options: Reference.

  **SAP HANA**: When you specify `PRESERVE_COL_NAMES=YES`, you can use reserved words for column names.

- To enable the DBMS to normalize the column names according to its naming conventions, specify variables using any case and set `PRESERVE_COLUMN_NAMES=NO`.

When you use SAS/ACCESS to read from, insert rows into, or modify data in an existing DBMS table, SAS identifies the database column names by their spelling. Therefore, when the database column exists, the case of the variable does not matter.

To save some time when coding, specify the `PRESERVE_NAMES=alias` if you plan to specify both the `PRESERVE_COL_NAMES= and `PRESERVE_TAB_NAMES= options in your LIBNAME statement.

To use column names in your SAS program that are not valid SAS names, you must use one of these techniques.

- Use the `DQUOTE=` option in PROC SQL and reference your columns using double quotation marks. Here is an example.

  ```sas
  proc sql dquote=ansi;
      select "Total$Cost" from mydblib.mytable;
  ```

- Specify the global system option `VALIDVARNAME=ANY` and use name literals in the SAS language. Here is an example.

  ```sas
  proc print data=mydblib.mytable;
      format 'Total$Cost'n 22.2;
  ```

If you are creating a table in PROC SQL, you must also include the `PRESERVE_COL_NAMES=YES` option in your LIBNAME statement. Here is an example.

```sas
libname mydblib oracle user=myusr1 password=mypwd1
    preserve_col_names=yes;
proc sql dquote=ansi;
    create table mydblib.mytable ("my$column" int);
```

---

**PRESERVE_GUID= LIBNAME Option**

Preserves the Microsoft SQL Server GUID (Unique Identifier).

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** NO
- **Data source:** OLE DB
Syntax

PRESERVE_GUID=NO | YES

**Syntax Description**

**NO**

retains the brackets that are around the GUID.

**YES**

removes the brackets from around the GUID.

**Details**

SAS/ACCESS to ODBC and SAS/ACCESS to OleDB delivers Microsoft SQL Server GUIDs (unique identifiers) differently.

- ODBC: 6F9619FF-8B86-D011-B42D-00C04FC964FF
- OLE DB: {6F9619FF-8B86-D011-B42D-00C04FC964FF}

The brackets must be removed from the GUIDs values before you execute a join. If the PRESERVE_GUID=NO option is set, the join fails. See the log below for PRESERVE_GUID=NO Option on page 188. Use the PRESERVE_GUID=YES option to remove the brackets. See the log for PRESERVE_GUID=YES Option on page 190.
Log 11.1  PRESERVE_GUID=NO Option

```sql
proc sql;
connect to oledb {
  init_string="Provider=SQLOLEDB.1;
  Password=secret;
  Persist Security Info=True;
  User ID=user_id;
  Initial Catalog=Pubs;
  data source=your_data_source"
};
execute (drop table t) by oledb;
execute (CREATE TABLE T(I INT, U uniqueidentifier)) by oledb;
execute (INSERT T VALUES(1, '6F9619FF-8B86-D011-B42D-00C04FC964FF')) by oledb;
disconnect from oledb;
quit;
NOTE: PROCEDURE SQL used (Total process time):
  real time           0.04 seconds
  cpu time            0.01 seconds

libname mylib oledb
  init_string="Provider=SQLOLEDB.1;
  Password=secret;
  Persist Security Info=True;
  User ID=user_id;
  Initial Catalog=Pubs;
  data source=your_data_source" preserve_guid=no ;
NOTE: Libref MYLIB was successfully assigned as follows:
  Engine:        OLEDB
  Physical Name:

/* if preserve_guid=no then the guid value should be in brackets {} */
ex: insert into mylib.t values (2, '{7F9619FF-8B86-D011-B42D-00C04FC964FF}');

if preserve_guid=yes then the guid value should not be in curly brackets {}
ex: insert into mylib.t values (2, '7F9619FF-8B86-D011-B42D-00C04FC964FF'); */

proc sql;
insert into mylib.t values (2, '{7F9619FF-8B86-D011-B42D-00C04FC964FF}');
NOTE: 1 row was inserted into MYLIB.t.
quit;
NOTE: PROCEDURE SQL used (Total process time):
  real time           0.01 seconds
  cpu time            0.00 seconds

proc print data=mylib.t;
run;
NOTE: There were 2 observations read from the data set MYLIB.t.
NOTE: PROCEDURE PRINT used (Total process time):
  real time           0.01 seconds
  cpu time            0.01 seconds
```
proc contents data=mylib.t;
run;

NOTE: PROCEDURE CONTENTS used (Total process time):
   real time     0.04 seconds
   cpu time      0.00 seconds

data x;
set mylib.t;
run;

NOTE: There were 2 observations read from the data set MYLIB.t.
NOTE: The data set WORK.X has 2 observations and 2 variables.
NOTE: DATA statement used (Total process time):
   real time     0.06 seconds
   cpu time      0.00 seconds

proc append data=x base=mylib.t;
run;

NOTE: Appending WORK.X to MYLIB.t.
NOTE: There were 2 observations read from the data set WORK.X.
NOTE: 2 observations added.
NOTE: The data set MYLIB.t has 4 observations and 2 variables.
NOTE: PROCEDURE APPEND used (Total process time):
   real time     0.06 seconds
   cpu time      0.01 seconds

proc print data=mylib.t;
run;

NOTE: There were 4 observations read from the data set MYLIB.t.
NOTE: PROCEDURE PRINT used (Total process time):
   real time     0.00 seconds
   cpu time      0.00 seconds
Log 11.2  PRESERVE_GUID=YES Option

```sql
proc sql;
  connect to oledb (
    init_string="
      Provider=SQLOLEDB.1;
      Password=secret;
      Persist Security Info=True;
      User ID=user_id;
      Initial Catalog=Pubs;
      data source=your_data_source"
  );

execute (drop table t) by oledb;
execute (CREATE TABLE T(I INT, U uniqueidentifier)) by oledb;
execute (INSERT T VALUES(1, '6F9619FF-8B86-D011-B42D-00C04FC964FF')) by oledb;
disconnect from oledb;
quit;
```

```
libname mylib oledb
  init_string="
    Provider=SQLOLEDB.1;
    Password=secret;
    Persist Security Info=True;
    User ID=user_id;
    Initial Catalog=Pubs;
    data source=your_data_source" preserve_guid=yes ;
```

```
/* if preserve_guid=no then the guid value should be in curly brackets {}
ex: insert into mylib.t values (2, '{7F9619FF-8B86-D011-B42D-00C04FC964FF}');
if preserve_guid=yes then the guid value should not be in curly brackets {}
ex: insert into mylib.t values (2, '7F9619FF-8B86-D011-B42D-00C04FC964FF'); */
```

```
proc sql;
  insert into mylib.t values (2, '7F9619FF-8B86-D011-B42D-00C04FC964FF');
```

```
quit;
```

```
proc print data=mylib.t;
run;
```

```
NOTE: There were 2 observations read from the data set MYLIB.t.
NOTE: PROCEDURE PRINT used (Total process time):
  real time 0.03 seconds
  cpu time 0.03 seconds
```
43 proc contents data=mylib.t;
44 run;

NOTE: PROCEDURE CONTENTS used (Total process time):
real time 0.06 seconds
cpu time 0.03 seconds

46 data x;
47 set mylib.t;
48 run;

NOTE: There were 2 observations read from the data set MYLIB.t.
NOTE: The data set WORK.X has 2 observations and 2 variables.
NOTE: DATA statement used (Total process time):
real time 0.07 seconds
cpu time 0.04 seconds

50 proc append data=x base=mylib.t;
51 run;

NOTE: Appending WORK.X to MYLIB.t.
NOTE: There were 2 observations read from the data set WORK.X.
NOTE: 2 observations added.
NOTE: The data set MYLIB.t has 4 observations and 2 variables.
NOTE: PROCEDURE APPEND used (Total process time):
real time 0.06 seconds
cpu time 0.03 seconds

53 proc print data=mylib.t;
54 run;

NOTE: There were 4 observations read from the data set MYLIB.t.
NOTE: PROCEDURE PRINT used (Total process time):
real time 0.01 seconds
cpu time 0.00 seconds

**PRESERVE_TAB_NAMES= LIBNAME Option**

Preserves spaces, special characters, and case sensitivity in DBMS table names.

**Valid in:** SAS/ACCESS LIBNAME statement

**Alias:** PRESERVE_NAMES= [see “Details”]

**Defaults:**
- YES [Microsoft SQL Server, MySQL, ODBC]
- NO [Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Netezza, OLE DB, Oracle, PC Files, PostgreSQL, SAP HANA, Sybase IQ, Teradata, Vertica]

**Interaction:** If you use the DS2 or FedSQL language, quoting and casing of names is different. For more information, see the identifiers topic in *SAS DS2 Language Reference* or *SAS FedSQL Language Reference*. 
Syntax

`PRESERVE_TAB_NAMES=YES | NO`

**Syntax Description**

**NO**

specifies that when you create DBMS tables or refer to an existing table, the table names are derived from SAS member names by using SAS member name normalization. However, the database applies DBMS-specific normalization rules to the SAS member names. Therefore, the table names are created or referenced in the database following the DBMS-specific normalization rules.

When you use SAS to read a list of table names (for example, in the SAS Explorer window), tables with names that do not conform to SAS member name normalization rules do not appear in output. In SAS line mode, here is how SAS indicates the number of tables that are not displayed from PROC DATASETS because of this restriction:

```
Due to the PRESERVE_TAB_NAMES=NO LIBNAME option setting, 12 table(s) have not been displayed.
```

You do not receive this warning when you use SAS Explorer. SAS Explorer displays DBMS table names in capitalized form when `PRESERVE_TAB_NAMES=NO`. This is now how the tables are represented in the DBMS.

NO is the default for most DBMS interfaces.

**YES**

specifies that table names are read from and passed to the DBMS with special characters, and the exact, case-sensitive spelling of the name is preserved.

*SAP HANA*: To use reserved words when naming a table for output to the database, you must specify `PRESERVE_TAB_NAMES=YES`.

**Details**

To use table names in your SAS program that are not valid SAS names, use one of these techniques.

- Use the PROC SQL option `DQUOTE=ansi` and place double quotation marks around the table name. The libref must specify `PRESERVE_TAB_NAMES=YES`. Here is an example.

```
libname mydblib oracle user=myusr1 password=mypwd1 preserve_tab_names=yes;
proc sql dquote=ansi;
   select * from mydblib."my table";
```

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: `PRESERVE_COL_NAMES= LIBNAME option, PRESERVE_TAB_NAMES= data set option, DBINDEX= data set option, SAS/ACCESS naming, SCHEMA= LIBNAME option`, naming conventions in the DBMS-specific reference section for your SAS/ACCESS interface.
• Use name literals in the SAS language. The libref must specify PRESERVE_TAB_NAMES=YES. Here is an example.

```sas
libname mydblib oracle user=myusr1
    password=mypwd1 preserve_tab_names=yes;
proc print data=mydblib.'my table'n;
run;
```

To save some time when coding, specify the PRESERVE_NAMES= alias if you plan to specify both the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options in your LIBNAME statement.

*Greenplum, HAWQ:* Unless you specify PRESERVE_TAB_NAMES=YES, the table name that you enter is converted to lowercase.

*Oracle:* Unless you specify PRESERVE_TAB_NAMES=YES, the table name that you enter for SCHEMA= LIBNAME option or for the DBINDEX= data set option is converted to uppercase.

**Example**

If you use PROC DATASETS to read the table names in an Oracle database that contains three tables, My_Table, MY_TABLE, and MY_TABLE. The results differ depending on the setting of PRESERVE_TAB_NAMES.

If the libref specifies PRESERVE_TAB_NAMES=NO, the PROC DATASETS output is one table name, MY_TABLE. This is the only table name that is in Oracle normalized form (uppercase letters and a valid symbol, the underscore). My_Table is not displayed because it is not in a form that is normalized for Oracle. MY TABLE is not displayed because it is not in SAS member normalized form: The embedded space is a nonstandard SAS character.

If the libref specifies PRESERVE_TAB_NAMES=YES, the PROC DATASETS output includes all three table names: My_Table, MY_TABLE, and MY TABLE.

---

**PRESERVE_USER= LIBNAME Option**

Preserves the case of the value in the USER= connection option.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** NO
- **Interactions:** Use the SAS_DB2_PRESERVE_USER environment variable for the entire SAS session.
  
  The LIBNAME option setting overrides the environment variable.

- **Data source:** DB2 under UNIX and PC Hosts
- **Note:** Support for this LIBNAME option was added in the first maintenance release for SAS 9.4.
- **Examples:** Set the LIBNAME option:

  ```sas
  libname mydblib db2 user=myuser1 pass=mypwd1 PRESERVE_USER=YES;
  ```

  Set the environment variable on PC hosts in the Advanced system settings:

  ```sas
  SAS_DB2_PRESERVE_USER YES
  ```

  Export the environment variable on UNIX hosts for the Bourne shell:

  ```bash
  PRESERVE_USER= LIBNAME Option
  ```
Export the environment variable on UNIX hosts for the C shell::

```bash
setenv SAS_DB2_PRESERVE_USER=YES
```

Set the environment variable at SAS invocation for UNIX and PC hosts:

```bash
sas -dms -set SAS_DB2_PRESERVE_USER YES
```

### Syntax

```plaintext
PRESERVE_USER=YES | NO
```

### Syntax Description

- **YES**
  - specifies that SAS/ACCESS preserves the case of the value for the USER= option.

- **NO**
  - specifies that SAS/ACCESS changes the value for the USER= option to uppercase for use in later connections.

### Details

DB2 typically accepts user names that are not case sensitive. However, certain authentication protocols such as LDAP allow case-sensitive user names. When you use such protocols with DB2, you can set `PRESERVE_USER=YES` so that the DB2 engine retains the original case of the USER= option.

### PROGRAM_NAME= LIBNAME Option

Specifies the string to use as the application identifier for DB2 monitoring.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Alias:** CORRELATION_ID
- **Default:** none
- **Data source:** DB2 under UNIX and PC Hosts
- **Note:** Support for this LIBNAME option was added for SAS 9.4.
- **Example:**

  ```sas
  LIBNAME db2data DB2=sample USER=db2 PWD=db2pwd PROGRAM_NAME='SAS on wrks1';
  ```

### Syntax

```plaintext
PROGRAM_NAME='user-defined-string'
```

### Details

The string that you specify overrides the default application identifier that DB2 chooses. If you are connecting to a host DB2 system through DB2 Connect, the first 12 characters of this string are used as the correlation ID on the host.
PROPERTIES= LIBNAME Option

Specifies JDBC custom connection properties, which override the default JDBC connection properties.

Valid in: SAS/ACCESS LIBNAME and CONNECT statements

Category: JDBC connection options

Default: none

Restriction: If you are using Cloudera Hadoop, you can specify only one JDBC connection property when HiveServer2 is prior to Hive 0.11.

Data source: Hadoop

Note: Support for this LIBNAME option was added in the second maintenance release for SAS 9.4.

Syntax

PROPERTIES= 'JDBC-connection-property-1';...'

Syntax Description

JDBC-connection-property
specifies one or more JDBC connection options to override the default JDBC connection options.

Details

When you specify JDBC connection properties using the PROPERTIES= LIBNAME option, the properties are appended to the JDBC URL. This overrides the default properties. Site-wide Hive properties are specified in the hive-site.xml file in the Hive configuration directory. In the JDBC URL, custom properties are separated from the default properties by the question mark ( ? ) character. The ? denotes the start of Hive configuration options. You do not need to add the ? character in the PROPERTIES= LIBNAME option. To specify Hive variables, add the character # before the Hive variable in the LIBNAME option.

Examples

Example 1: Set Strict Mode

libname h4 hadoop SUBPROTOCOL=HIVE2 schema=sample user=hdusr1 server="hdp2ga"
properties='hive.mapred.mode=strict';
jdbc:hive2://hdp2ga.unx.sas.com:10000?hive.mapred.mode=strict

Example 2: Set Strict Mode with a Second Option

libname h4 hadoop SUBPROTOCOL=HIVE2 schema=sample user=hdusr1 server="hdp2ga"
properties='hive.mapred.mode=strict;hive.optimize.groupby=false';
jdbc:hive2://hdp2ga.unx.sas.com:10000?hive.mapred.mode=strict;
hive.optimize.groupby=false
Example 3: Set a Hive Variable

libname h4 hadoop SUBPROTOCOL=HIVE2 schema=sample user=hdusr1 server="hdp2ga"
    properties='#D_TBL=dummy_t';
jdbc:hive2://hdp2ga.unx.sas.com:10000?#D_TBL=dummy_t

Example 4: Set Strict Mode and a Hive Variable

libname h4 hadoop SUBPROTOCOL=HIVE2 schema=sample user=hdusr1 server="hdp2ga"
    properties='hive.mapred.mode=strict#D_TBL=dummy_t';
jdbc:hive2://hdp2ga.unx.sas.com:10000?hive.mapred.mode=strict#D_TBL=dummy_t

Example 5: Set Strict Mode and a Hive Principal

libname h4 hadoop SUBPROTOCOL=HIVE2 principal=hive/HiveServer2Host@YOUR-REALM.COM
    schema=sample user=hdusr1 server="hdp2ga"
    properties='hive.mapred.mode=strict#D_TBL=dummy_t';
jdbc:hive2://hdp2ga.unx.sas.com:10000;
principal=
hive/HiveServer2Host@YOUR-REALM.COM?hive.mapred.mode=strict#D_TBL=dummy_t

QUALIFIER= LIBNAME Option

Allows identification of such database objects as tables and views with the specified qualifier.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Data source: Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Vertica
See: QUALIFIER= data set option

Syntax

QUALIFIER=<qualifier-name>

Details

If you omit this option, the default is the default DBMS qualifier name, if any. You can use QUALIFIER= for any DBMS that allows three-part identifier names, such as qualifier.schema.object.

MySQL: The MySQL interface does not support three-part identifier names, so a two-part name is used (such as qualifier.object).

Example

In this LIBNAME statement, the QUALIFIER= option causes ODBC to interpret any reference to mydblib.employee in SAS as mydept.scott.employee.

    libname mydblib odbc dsn=myoracle
        password=testpass schema=scott
        qualifier=mydept;

In this example, the QUALIFIER= option causes OLE DB to interpret any reference in SAS to mydblib.employee as pcdivision.raoul.employee.

    libname mydblib oledb provider=SQLOLEDB
QUALIFY_ROWS= LIBNAME Option
Uniquely qualifies all member values in a result set.

Valid in: SAS/ACCESS LIBNAME statement
Default: NO
Data source: OLE DB
See: Accessing OLE DB for OLAP Data

Syntax
QUALIFY_ROWS=YES | NO

Syntax Description
YES
specifies that when the OLE DB interface flattens the result set of an MDX command, the values in each column are uniquely identified using a hierarchical naming scheme.

NO
specifies that when the OLE DB interface flattens the result set of an MDX command, the values in each column are not qualified, which means they might not be unique.

Details
For example, when this option is set to NO, a GEOGRAPHY column might have a value of PORTLAND for Portland, Oregon, and the same value of PORTLAND for Portland, Maine. When you set this option to YES, the two values might become [USA].[Oregon].[Portland] and [USA].[Maine].[Portland], respectively.

Note: Depending on the size of the result set, QUALIFY_ROWS=YES can have a significant, negative impact on performance. This can occur because it forces the OLE DB interface to search through various schemas to gather the information needed to create unique qualified names.

QUERY_BAND= LIBNAME Option
Specifies whether to set a query band for the current session.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Requirement: A semi-colon (;) is required before the ending quotation mark in order for the database to process "pair-name=pair_value;"
### QUERY_BAND= LIBNAME Option

Specifies the number of seconds of inactivity to wait before canceling a query.

**Syntax**

```plaintext
QUERY_BAND="pair-name=pair_value;"
```

**Syntax Description**

`pair-name=pair_value`

specifies a name and value pair of a query band for the current session.

**Details**

Use this option to set unique identifiers in Teradata sessions and to add them to the current session. The Teradata engine uses this syntax to pass the name-value pair to Teradata:

```plaintext
libname db teradata user=myusr1 password=mypwd1
QUERY_BAND="org=Marketing;report=Mkt4Q08;";
```

For more information about this option and query-band limitations, see *Teradata SQL Reference: Data Definition Statements*.

---

### QUERY_TIMEOUT= LIBNAME Option

Specifies the number of seconds of inactivity to wait before canceling a query.

**Valid in:** SAS/ACCESS LIBNAME, CONNECT statement

**Default:** 0

**Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, PostgreSQL, Sybase IQ, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** QUERY_TIMEOUT= data set option

**Syntax**

```plaintext
QUERY_TIMEOUT=number-of-seconds
```

**Syntax Description**

`number-of-seconds`

specifies a positive integer for the number of seconds to wait before canceling the query. The default value of 0 indicates that there is no time limit for a query. This option is useful when you are testing a query or if you suspect that a query might contain an endless loop.
QUOTE_CHAR= LIBNAME Option

Specifies which quotation mark character to use when delimiting identifiers.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Data source: Aster, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase IQ, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

Syntax
QUOTE_CHAR=character

Syntax Description
character
the quotation mark character to use when delimiting identifiers, such as the double quotation mark (").

Details
The provider usually specifies the delimiting character. However, when there is a difference between what the provider and the DBMS allow for this character, the QUOTE_CHAR= option overrides the character that the provider returns.

Microsoft SQL Server: QUOTE_CHAR= overrides the Microsoft SQL Server default.

ODBC: This option is mainly for the ODBC interface to Sybase, and you should use it with the DBCONINIT and DBLIBINIT LIBNAME options. QUOTE_CHAR= overrides the ODBC default because some drivers return a blank for the identifier delimiter even though the DBMS uses a quotation mark (for example, ODBC to Sybase).

Examples

Example 1: Specify a Single Quotation Mark
Here is what to specify if you want your quotation character to be a single quotation mark.

libname x odbc dsn=mydsn pwd=mypassword quote_char=''';

Example 2: Specify a Double Quotation Mark
Here is what to specify if you want your quotation character to be a double quotation mark.

libname x odbc dsn=mydsn pwd=mypassword quote_char='"';
Syntax

\texttt{QUOTED_{IDENTIFIER}=YES | NO}

Details

You use this option in place of the \texttt{PRESERVE\_COL\_NAMES=} and \texttt{PRESERVE\_TAB\_NAMES=} \texttt{LIBNAME} options. They have no effect on the Sybase interface because it defaults to case sensitivity.

\textbf{READBUFF=} \texttt{LIBNAME} Option

Specifies the number of rows of DBMS data to read into the buffer.

- **Valid in:** SAS/ACCESS \texttt{LIBNAME} statement, \texttt{CONNECT} statement
- **Aliases:** \texttt{BUFFSIZE=} [Oracle] \texttt{ROWSET\_SIZE=} [Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, Sybase, Sybase IQ] \texttt{ROWSET=} [Aster, Greenplum, HAWQ, Sybase IQ]
- **Default:** DBMS-specific
- **Restriction:** When \texttt{READBUFF=}1, only one row is retrieved at a time.
- **Interaction:** Buffering data reads can decrease network activities and increase performance. However, because SAS stores the rows in memory, higher values for \texttt{READBUFF=} use more memory. In addition, if too many rows are selected at once, rows that are returned to the SAS application might be out of date. For example, if someone else modifies the rows, you do not see the changes.
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **Tips:** This option improves performance by specifying a number of rows that can be held in memory for input into SAS. The higher the value for \texttt{READBUFF=} , the more rows that the engine retrieves in one fetch operation.
- **See:** \texttt{INSERTBUFF=} \texttt{LIBNAME} option, \texttt{INSERTBUFF=} data set option, \texttt{READBUFF=} data set option

**Syntax**

\texttt{READBUFF=} \texttt{integer}
Syntax Description

integer

the positive number of rows to hold in memory. SAS allows the maximum number that the DBMS allows.

Details

Table 11.6  DBMS-Specific Default Values

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aster, Greenplum, HAWQ, Impala, Netezza, SAP HANA, Sybase IQ</td>
<td>The default is automatically calculated based on row length.</td>
</tr>
<tr>
<td>DB2 under UNIX and PC Hosts</td>
<td>If you do not specify a value for this option, the default buffer size is automatically calculated based on the row length of your data. The SQLExtendedFetch API call is used.</td>
</tr>
<tr>
<td>DB2 under z/OS</td>
<td>For SAS 9.2 and above, the default is 1 and the maximum value is 32,767.</td>
</tr>
<tr>
<td>Microsoft SQL Server, ODBC</td>
<td>The default is 0. If you do not specify a value for this option, the SQLFetch API call is used and no internal SAS buffering is performed. When you set READBUFF=1 or greater, the SQLExtendedFetch API call is used.</td>
</tr>
<tr>
<td>OLE DB, Vertica</td>
<td>The default is 1.</td>
</tr>
<tr>
<td>Oracle</td>
<td>The default is 250. If you do not specify this option, the READBUFF= value is automatically calculated as the number of rows that can fit into a memory buffer of 100K on most platforms or 50K on the z/OS platform. However, it is adjusted to be within the range of 250–15000. You can always override the default value by explicitly specifying a value for this option.</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>The default is 0.</td>
</tr>
<tr>
<td>Sybase</td>
<td>The default is 100. To use a value greater than 1 row for this option, you must set CONNECTION=UNIQUE.</td>
</tr>
</tbody>
</table>

READ_ISOLATION_LEVEL= LIBNAME Option

Defines the degree of isolation of the current application process from other concurrently running application processes.

Valid in: SAS/ACCESS LIBNAME statement

Default: not set [OLE DB, Oracle], COMMITTED READ [Informix], determined by the DBMS [DB2 under z/OS], 1 [Sybase], RC [Greenplum, HAWQ, Microsoft SQL Server, ODBC, PostgreSQL, Sybase IQ, Vertica]. You can set the default in the DB2Cli.ini file [DB2 under UNIX and PC Hosts, ODBC]
### Requirement:
For DB2 under UNIX and PC Hosts and ODBC, this option is ignored if you do not set the `READ_LOCK_TYPE=` LIBNAME option to ROW.

### Data source:
DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, ODBC, OLE DB, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica

### Note:
Support for HAWQ was added in the third maintenance release for SAS 9.4.

### See:
- `CONNECTION= LIBNAME option`, `READ_ISOLATION_LEVEL= data set option`, `READ_LOCK_TYPE= LIBNAME option`, `READ_LOCK_TYPE= data set option`, `UPDATE_ISOLATION_LEVEL= LIBNAME option`, `UPDATE_ISOLATION_LEVEL= data set option`, `UPDATE_LOCK_TYPE= LIBNAME option`, `UPDATE_LOCK_TYPE= data set option`, and DBMS-specific locking information in the reference section for your SAS/ACCESS interface

---

### Syntax

**READ_ISOLATION_LEVEL=** `DBMS-specific value`

### Details

The degree of isolation defines the degree to which these items are affected:

- Rows that the current application reads and updates are available to other concurrently executing applications.
- Update activity of other concurrently executing application processes can affect the current application.

---

### READ_LOCK_TYPE= LIBNAME Option

Specifies how data in a DBMS table is locked during a READ transaction.

**Valid in:** SAS/ACCESS LIBNAME statement

**Default:** none [DB2 under z/OS, Teradata], set by the data provider [OLE DB], NOLOCK [Oracle, Sybase], ROW [DB2 under UNIX and PC Hosts, Microsoft SQL Server, ODBC, PostgreSQL, Sybase IQ, Vertica]

**Data source:** DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, OLE DB, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica

**Tip:** If you omit `READ_LOCK_TYPE=`, the default is the default action for the DBMS. You can set a lock for one DBMS table by using the data set option or for a group of DBMS tables by using the LIBNAME option.

**See:** `CONNECTION= LIBNAME option`, `READ_ISOLATION_LEVEL= LIBNAME option`, `READ_ISOLATION_LEVEL= data set option`, `READ_LOCK_TYPE= data set option`, `UPDATE_ISOLATION_LEVEL= LIBNAME option`, `UPDATE_ISOLATION_LEVEL= data set option`, `UPDATE_LOCK_TYPE= LIBNAME option`, `UPDATE_LOCK_TYPE= data set option`, and DBMS-specific locking information in the reference section for your SAS/ACCESS interface

---

### Syntax

**READ_LOCK_TYPE=** `ROW | PAGE | TABLE | NOLOCK | VIEW`
Syntax Description

ROW
locks a row if any of its columns are accessed. If you are using the interface to ODBC or DB2 under UNIX and PC Hosts, READ_LOCK_TYPE=ROW indicates that locking is based on the READ_ISOLATION_LEVEL= LIBNAME option.

Data source: DB2 under UNIX and PC Hosts, Microsoft SQL Server, ODBC, Oracle, PostgreSQL, Sybase IQ, Vertica

PAGE
locks a page of data, which is a DBMS-specific number of bytes.

Data source: Sybase

TABLE
locks the entire DBMS table. If you specify READ_LOCK_TYPE=TABLE, you must also specify CONNECTION=UNIQUE, or you receive an error message. Setting CONNECTION=UNIQUE ensures that your table lock is not lost (for example, due to another table closing and committing rows in the same connection).

Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, Oracle, Sybase IQ, Teradata

NOLOCK
does not lock the DBMS table, pages, or rows during a read transaction.

Data source: Microsoft SQL Server, ODBC with Microsoft SQL Server driver, OLE DB, Oracle, Sybase

VIEW
locks the entire DBMS view.

Data source: Teradata

Example

In this example, the libref MYDBLIB uses SAS/ACCESS Interface to Oracle to connect to an Oracle database. USER=, PASSWORD=, and PATH= are SAS/ACCESS connection options. The LIBNAME options specify to use row-level locking when data is read or updated.

```
libname mydblib oracle user=myusr1 password=mypwd1 path=mysrv1 read_lock_type=row update_lock_type=row;
```
Syntax

READ_METHOD=JDBC | HDFS

Syntax Description

JDBC
specifies that data is to be read through the JDBC connection to the Hive service.

HDFS
specifies that data is to be read through a connection to the Hadoop HDFS service.

Details

Although HDFS cannot alter the behavior of operations that always use JDBC, in general HDFS is a faster alternative to JDBC. To take advantage of potential performance benefits, set this option to HDFS. Use JDBC when you cannot access the HDFS service or JDBC Read offers some other advantage.

Example: Read Data Using JDBC

In this example, a partition of data from the sales Hive table is read using JDBC.

```sas
libname hdp hadoop server=mysrv1 user=myusr1 pwd=mypwd1;
data work.sales_subset; set hdp.sales(READ_METHOD=JDBC);
where year_month='2012-10'; run;
```

---

READ_MODE_WAIT= LIBNAME Option

During SAS/ACCESS Read operations, specifies whether Teradata should wait to acquire a lock or fail the request when a different user has already locked the DBMS resource.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Data source:</td>
<td>Teradata</td>
</tr>
<tr>
<td>See:</td>
<td>READ_MODE_WAIT= data set option, Locking in the Teradata Interface</td>
</tr>
</tbody>
</table>

Syntax

READ_MODE_WAIT= YES | NO

Syntax Description

YES
specifies for Teradata to wait to acquire the lock, so SAS/ACCESS waits indefinitely until it can acquire the lock.

NO
specifies Teradata fails the lock request if the specified DBMS resource is locked.
Details

If you specify READ_MODE_WAIT=NO and if a different user holds a restrictive lock, the executing SAS step fails. SAS/ACCESS continues processing the job by executing the next step.

If you specify READ_MODE_WAIT=YES, SAS/ACCESS waits indefinitely until it can acquire the lock.

A restrictive lock means that another user is holding a lock that prevents you from obtaining the lock that you want. Until the other user releases the restrictive lock, you cannot obtain your lock. For example, another user's table-level WRITE lock prevents you from obtaining a READ lock on the table.

REMOTE_DBTYPE= LIBNAME Option

Specifies whether the libref points to a database server on z/OS or to one on Linux, UNIX, or Windows.

Valid in: SAS/ACCESS LIBNAME statement
Default: ZOS
Restriction: This option is ignored if you do not use is with either the SERVER= CONNECT statement option or the LOCATION= LIBNAME option.
Requirement: Use this option with the SERVER= CONNECT statement option or the LOCATION= LIBNAME option.
Data source: DB2 under z/OS
See: LOCATION= LIBNAME option, SERVER= CONNECT statement option (SQL Pass-Through Facility Specifics for DB2 under z/OS - Key Information)

Syntax

REMOTE_DBTYPE=LUW | ZOS

Syntax Description

LUW
specifies that the database server that is accessed through the libref resides on Linux, UNIX, or Windows (LUW).

ZOS
specifies that the database server that is accessed through the libref resides on z/OS.

Details

Specifying REMOTE_DBTYPE= in the LIBNAME statement ensures that the SQL that some SAS procedures use to access the DB2 catalog tables is generated properly and is based on the database server type. It also lets such special catalog calls as DBMS::Indexes function properly when the target database does not reside on a mainframe computer.

If the target data source is a DB2 LUW or another DB2 database on z/OS, the SQL dictionary is loaded when you specify this option.
Example

This example uses REMOTE_DBTYPE= with the SERVER= option.

libname mylib db2 ssid=db2a server=db2_udb remote_dbtype=luw;
proc datasets lib=mylib;
quit;

By specifying REMOTE_DBTYPE=LUW, this SAS code lets the catalog call work properly for this remote connection.

proc sql;
   connect to db2 (ssid=db2a server=db2_udb remote_dbtype=luw);
   select * from connection to db2
   select * from connection to db2
   {DBMS::PrimaryKeys ("", "JOSMITH", ")");
quit;

REREAD_EXPOSURE= LIBNAME Option

Specifies whether the SAS/ACCESS engine functions like a random access engine for the scope of the LIBNAME statement.

| Valid in: | SAS/ACCESS LIBNAME statement |
| Default: | NO |
| Data source: | Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica |
| Note: | Support for HAWQ was added in the third maintenance release for SAS 9.4. |
| See: | UPDATE_ISOLATION_LEVEL= LIBNAME option, UPDATE_LOCK_TYPE= LIBNAME option |

Syntax

REREAD_EXPOSURE=YES | NO

Syntax Description

NO
 specifies that the SAS/ACCESS engine functions as an RMOD engine, which means that your data is protected by the normal data protection that SAS provides.

YES
 specifies that the SAS/ACCESS engine functions like a random access engine when rereading a row so that you cannot guarantee that the same row is returned. For example, if you read row 5 and someone else deletes it, you read a different row the next time you read row 5. You have the potential for data integrity exposures within the scope of your SAS session.

Details

CAUTION:

Using REREAD_EXPOSURE= could cause data integrity exposures.
Netezza, ODBC, OLE DB: If you set this option to YES, it is advisable to set UPDATE_ISOLATION_LEVEL=S (serializable) to avoid data integrity problems.

Oracle: If you set this option to YES, it is advisable to set UPDATE_LOCK_TYPE=TABLE to avoid data integrity problems.

**RESULTS= LIBNAME Option**

Determines where to store query results.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>MEMORY</td>
</tr>
<tr>
<td>Data source:</td>
<td>MySQL</td>
</tr>
</tbody>
</table>

**Syntax**

RESULTS=MEMORY | SERVER | DISK

**Syntax Description**

- **MEMORY**
  - stores query results in client memory.
- **SERVER**
  - stores query results on the server.
- **DISK**
  - stores query results in a temporary disk file on the client computer.

**Details**

Multiple concurrent connections to the server are not supported. Therefore, when RESULTS=SERVER, the entire query must be one that you can push to the server. If not, this message appears in the SAS log:

Commands out of sync; you can't run this command now.

RESULTS=DISK lets you run complex queries with result sets that would typically cause an out-of-memory error. Result-set size is limited only to the free space on the drive that is used for temporary files.

**Example: Sending Results to a Temporary Disk File**

    libname spooled mysql...results=disk;

**SAS_DBMS_AUTOMETADATA= LIBNAME Option**

Specifies whether to transfer data types and character-set metadata from input to output for DATA step processing.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>NO</td>
</tr>
</tbody>
</table>
Interaction: Use the SAS_DBMS_AUTOMETADATA environment variable for the entire SAS session.

Data source: Teradata

Note: Support for this LIBNAME option was added for SAS 9.4.

Examples: Set the LIBNAME option:

libname tdlib teradata user=myuser1 pass=mypwd1 SAS_DBMS_AUTOMETADATA=YES;

Set the environment variable on PC hosts:

SAS_DBMS_AUTOMETADATA 1

Export the environment variable on UNIX hosts [Bourne and Korn shells]:

SAS_DBMS_AUTOMETADATA=1
export SAS_DBMS_AUTOMETADATA

Export the environment variable on UNIX hosts [C shell]:

export SAS_DBMS_AUTOMETADATA=1

Set the environment variable at SAS invocation for PC, UNIX, and z/OS hosts:

sas -dms -set SAS_DBMS_AUTOMETADATA 1

Syntax

SAS_DBMS_AUTOMETADATA=YES | NO

SCHEMA= LIBNAME Option

Allows reading of such database objects as tables and views in the specified schema.

Valid in: SAS/ACCESS LIBNAME statement

Alias: DATABASE= [Impala, Teradata]

Default: DBMS-specific

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Notes: Support for Netezza was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: SCHEMA= data set option, PRESERVE_TAB_NAMES= LIBNAME option

Syntax

SCHEMA= schema-name

Syntax Description

schema-name specifies the name that is assigned to a logical classification of objects in a relational database.
Details

For this option to work, you must have the appropriate privileges to the specified schema.

If you do not specify this option, you connect to the default schema for your DBMS.

The values for SCHEMA= are typically case sensitive, so make sure that the value that you specify exactly matches the case of the schema name for your database.

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aster</td>
<td><code>none</code></td>
</tr>
<tr>
<td></td>
<td>This uses the database user's default schema. However, the user name is used instead when the user's default scheme is the user name. An example is when SQLTables is called to obtain a table listing using PROC DATASETS or SAS Explorer.</td>
</tr>
<tr>
<td>DB2</td>
<td><code>your user ID</code></td>
</tr>
<tr>
<td>Greenplum, HAWQ, Impala, Microsoft SQL Server, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase IQ, Vertica</td>
<td><code>none</code>.</td>
</tr>
<tr>
<td></td>
<td>Note: For PostgreSQL, the schema name cannot be the same as a user ID.</td>
</tr>
<tr>
<td>Informix</td>
<td><code>your user name</code></td>
</tr>
<tr>
<td></td>
<td>Note: The SCHEMA LIBNAME option is ignored when using implicit pass-through and the Informix engine.</td>
</tr>
<tr>
<td>Netezza</td>
<td><code>your default schema</code></td>
</tr>
<tr>
<td>Oracle</td>
<td><code>Specify a schema name to be used when referring to database objects. SAS can access another user's database objects by using a specified schema name. If PRESERVE_TAB_NAMES=NO, SAS converts the SCHEMA= value to uppercase because all values in the Oracle data dictionary are uppercase unless quoted.</code></td>
</tr>
<tr>
<td>Sybase</td>
<td><code>none</code></td>
</tr>
<tr>
<td></td>
<td>You cannot use the SCHEMA= option when you use UPDATE_LOCK_TYPE=PAGE to update a table.</td>
</tr>
<tr>
<td>Teradata</td>
<td><code>If you omit this option, a libref points to your default Teradata database, which often has the same name as your user name. You can use this option to point to a different database. This option lets you view or modify another user's DBMS tables or views if you have the required Teradata privileges. (For example, to read another user's tables, you must have the Teradata privilege SELECT for that user's tables.) For more information about changing the default database, see the DATABASE statement in your Terada documentation.</code></td>
</tr>
</tbody>
</table>
Example

In this example, SCHEMA= causes DB2 to interpret any reference in SAS to mydb.employee as scott.employee.

libname mydb db2 SCHEMA=SCOTT;

To access an Oracle object in another schema, use the SCHEMA= option, as in this example. The schema name is typically a user name or ID.

libname mydblib oracle user=myusr1 password=mypwd1 path='mysrv1' schema=john;

In this example, the Oracle SCHEDULE table resides in the AIRPORTS schema and is specified as AIRPORTS.SCHEDULE. To access this table in PROC PRINT and still use the libref (CARGO) in the SAS/ACCESS LIBNAME statement, specify the schema in the SCHEMA= option. Then put in the libref.table the DATA statement for the procedure.

libname cargo oracle schema=airports user=myusr1 password=mypwd1 path="mysrv1";
proc print data=cargo.schedule;
run;

In this Teradata example, the MYUSR1 user prints the emp table, which is located in the other user database.

libname mydblib teradata user=myusr1 pw=mypwd1 schema=otheruser;
proc print data=mydblib.emp;
run;

SECTIONS= LIBNAME Option

Specifies how many Teradata sessions to be logged on when using FastLoad, FastExport, or Multiload.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Data source: Teradata
See: FASTEXPORT= LIBNAME option, SESSIONS= data set option, Using MultiLoad

Syntax

SESSIONS=number-of-sessions

Syntax Description

number-of-sessions
specifies a numeric value that indicates the number of sessions to be logged on.

Details

When you read data with FastExport or load data with FastLoad or MultiLoad, you can request multiple sessions to increase throughput. Using large values might not necessarily increase throughput due to the overhead associated with session management. Check whether your site has any recommended value for the number of
sessions to use. See your Teradata documentation for details about using multiple sessions.

Example

This example uses SESSIONS= in a LIBNAME statement to request five sessions for loading data with FastLoad.

```sas
libname x teradata user=myusr1 pw=mypwd1 SESSIONS=5;
proc datasets library=x;
  delete test;run;
data x.test(FASTLOAD=YES);
i=5;
run;
```

SHOW_SYNONYMS= LIBNAME Option

Specifies whether PROC DATASETS shows synonyms, tables, views, or materialized views for the current user and schema if you specified the SCHEMA= option.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>YES</td>
</tr>
<tr>
<td>Data source:</td>
<td>Oracle</td>
</tr>
<tr>
<td>See:</td>
<td>DBLINK= LIBNAME option, DB_OBJECTS= LIBNAME option, SCHEMA= LIBNAME option, SCHEMA= data set option</td>
</tr>
</tbody>
</table>

Syntax

SHOW_SYNONYMS=**YES** | NO

Syntax Description

**YES**

specifies that PROC DATASETS shows only synonyms that represent tables, views, or materialized views for the current user.

**NO**

specifies that PROC DATASETS shows only tables, views, or materialized views for the current user.

Details

Rather than submit PROC DATASETS, you can select the libref in SAS Explorer to obtain this same information. By default, no PUBLIC synonyms are displayed unless you specify SCHEMA=PUBLIC.

When you specify only the SCHEMA option, the current schema is always displayed with the appropriate privileges.

Tables, views, materialized views, or synonyms on the remote database always are displayed when you specify the DBLINK=LIBNAME option. If a synonym represents
an object on a remote database that you might not be able to read, you might receive an Oracle error. An example is a synonym representing a sequence.

Synonyms, tables, views, and materialized views in a different schema are also displayed.

---

**SLEEP= LIBNAME Option**

Specifies the number of minutes that FastExport, FastLoad, or MultiLoad waits before trying again to log on to Teradata.

**Valid in:** SAS/ACCESS LIBNAME statement

**Default:** 6

**Data source:** Teradata

**Tip:** The data set option has precedence over the LIBNAME option.

**See:** DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option, DBSLICEPARM= system option, FASTEXPORT= LIBNAME option, Maximizing Teradata Load Performance, MULTILOAD= data set option, SLEEP= data set option, TENACITY= LIBNAME option, TENACITY= data set option, Using the TPT API

---

**Syntax**

SLEEP=**number-of-minutes**

**Syntax Description**

*number-of-minutes*  
the number of minutes to wait before trying again to log on to Teradata.

**Details**

Use this option to indicate to FastExport, FastLoad, or FastLoadMultiLoad how long to wait before trying to log on to Teradata again when the maximum number of utilities are already running. (The maximum number of Teradata utilities that can run concurrently varies from 5 to 15, depending on the database server setting.) The default value for SLEEP= is 6 minutes. The value that you specify for SLEEP= must be greater than 0.

Use SLEEP= with TENACITY=. TENACITY= specifies the time in hours that FastExport, FastLoad, or MultiLoad must continue to try the logon operation. SLEEP= and TENACITY= function very much like the SLEEP and TENACITY run-time options of the native Teradata FastExport, FastLoad, or MultiLoad utility.

---

**SPOOL= LIBNAME Option**

Specifies whether SAS creates a utility spool file during read transactions that read data more than once.

**Valid in:** SAS/ACCESS LIBNAME statement

**Default:** YES

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: CONNECTION= LIBNAME option

Syntax
SPOOL=YES | NO | DBMS

Syntax Description
YES
specifies that SAS creates a utility spool file into which it writes the rows that are
read the first time. For subsequent passes through the data, the rows are read from
the utility spool file rather than being reread from the DBMS table. This guarantees
that the rowset is the same for every pass through the data.

NO
specifies that the required rows for all passes of the data are read from the DBMS
table. No spool file is written. There is no guarantee that the rowset is the same for
each pass through the data.

DBMS [valid only for Oracle]
specifies that the required rows for all passes of the data are read from the DBMS
table. However, additional enforcements are made on the DBMS server side to
ensure that the rowset is the same for every pass through the data. This setting causes
SAS/ACCESS Interface to Oracle to satisfy the two-pass requirement by starting a
read-only transaction. SPOOL=YES and SPOOL=DBMS have comparable
performance results for Oracle. However, SPOOL=DBMS does not use any disk
space. When SPOOL is set to DBMS, you must set CONNECTION=UNIQUE or an
error occurs.

Details
In some cases, SAS processes data in more than one pass through the same set of rows.
Spooling is the process of writing rows that have been retrieved during the first pass of a
data read to a spool file. In the second pass, rows can be reread without performing input
and output to the DBMS a second time. When data must be read more than once,
spooling improves performance. Spooling also guarantees that the data remains the same
between passes, as most SAS/ACCESS interfaces do not support member-level locking.

MySQL: Do not use SPOOL=NO with the MySQL interface.

Teradata: SPOOL=NO requires SAS/ACCESS to issue identical SELECT statements to
Teradata twice. In addition, because the Teradata table can be modified between passes,
SPOOL=NO can cause data integrity problems. Use SPOOL=NO with discretion.

SQL_FUNCTIONS= LIBNAME Option
Customizes the in-memory SQL dictionary function list for this particular LIBNAME statement.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Restrictions: Informix and OLE DB support only SQL_FUNCTIONS=ALL.
You must specify a two-part data set name, such as <libref.member> or an error
results.
<libref.member> must be a SAS data set. No check is performed to ensure that it is assigned to the default Base SAS engine.

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: SQL_FUNCTIONS_COPY= LIBNAME option, also the “Passing SAS Functions” section for your specific SAS/ACCESS interface

Syntax

SQL_FUNCTIONS=ALL <libref.member> | EXTERNAL_REPLACE=<libref.member> | EXTERNAL_APPEND=<libref.member>

Syntax Description

ALL

customizes the in-memory SQL dictionary function list for this particular LIBNAME statement by adding the set of all existing functions, even those that might be risky or untested.

EXTERNAL_REPLACE=<libref.member>

indicates a user-specified, external SAS data set from which the complete function list in the SQL dictionary is to be built. The assumption is that the user has already issued a LIBNAME statement to the directory where the SAS data set exists.

Restriction This value is not valid for Informix or OLE DB.

EXTERNAL_APPEND=<libref.member>

indicates a user-specified, external SAS data set from which additional functions are to be added to the existing function list in the SQL dictionary. The assumption is that the user has already issued a LIBNAME statement to the directory where the SAS data set exists.

Restriction This value is not valid for Informix or OLE DB.

Details

Using this option can cause unexpected results, especially if you use it for NULL processing and for handling date, time, and timestamp. For example, when executed without SQL_FUNCTIONS= enabled, this SAS code returns the SAS date 15308.

```sas
proc sql;
  select distinct DATE () from x.test;
quit;
```

However, with SQL_FUNCTIONS=ALL, the same code returns 2001-1-29, which is an ODBC date format. So you should exercise care when you use this option.

Functions that are passed are different for each DBMS. See the DBMS-specific reference section for your SAS/ACCESS interface for list of functions that it supports.

Here are additional details to keep in mind when you add to or modify the SAS data set.
Variable | Required* | Optional** | Read-Only*** | Valid Values
--- | --- | --- | --- | ---
SASFUNCNAME | X | | | Truncated to 32 characters if length is greater than 32
SASFUNCNAMELEN | X | | | Must correctly reflect the length of SASFUNCNAME
DBMSFUNCNAME | X | | | Truncated to 50 characters if length is greater than 50
DBMSFUNCNAMELEN | X | | | Must correctly reflect the length of DBMSFUNCNAME
FUNCTION_CATEGORY | X | | | AGGREGATE, CONSTANT, SCALAR
FUNC_USAGE_CONTEXT | X | | | SELECT_LIST, WHERE_ORDERBY
FUNCTION_RETURNTYPE | X | | | BINARY, CHAR, DATE, DATETIME, DECIMAL, GRAPHIC, INTEGER, INTERVAL, NUMERIC, TIME, VARCHAR
FUNCTION_NUM_ARGS | X | | | 0
CONVERT_ARGS | | X | | Must be set to 0 for a newly added function.
ENGINEINDEX | | X | | Must remain unchanged for existing functions. Set to 0 for a newly added function.

* An error results when a value is missing.
** For new and existing functions.

### Examples

**Example 1: Include and Replace Existing Functions**

You can use EXTERNAL_APPEND= to include one or more existing functions to the in-memory function list and EXTERNAL_REPLACE= to replace them. In this example, the DATEPART function in a SAS data set of Oracle functions by appending the function to an existing list of SAS functions.

```sql
proc sql;
create table work.append as select * from work.allfuncs where sasfuncname='DATEPART';
quit;
libname mydblib oracle sql_functions='EXTERNAL_APPEND=work.append'
sql_functions_copy=saslog;
```

**Example 2: Replace All SAS Functions with the Oracle Equivalent**

In this example, the equivalent Oracle functions in a SAS data set replace all SAS functions that contain the letter I.

```sql
proc sql;
create table work.replace as select *
```
from work.allfuncs where sasfuncname like '%I%';
quit;
libname mydblib oracle sql_functions="EXTERNAL_REPLACE=work.replace"
   sql_functions_copy=saslog;

Example 3: Add a New Function

data work.newfunc;
   SASFUCNNAME = "sasname";
   SASFUCNNAMELEN = 7;
   DBMSFUCNNAME = "DBMSUDFName";
   DBMSFUCNNAMELEN = 11;
   FUNCTION_CATEGORY = "CONSTANT";
   FUNC_USAGE_CONTEXT = "WHERE_ORDERBY";
   FUNCTION_RETURNTYP = "NUMERIC";
   FUNCTION_NUM_ARGS = 0;
   CONVERT_ARGS = 0;
   ENGINEINDEX = 0;
output;
runk;

/* Add function to existing in-memory function list */
libname mydblib oracle sql_functions="EXTERNAL_APPEND=work.newfunc"
   sql_functions_copy=saslog;

Example 4: Add and Run the Netezza Variant of the SOUNDEX Function

Netezza supports the NYSIIS function, which is a variant of the SOUNDEX function and produces similar results. This example shows how to specify that NYSIIS should be called when the SOUNDEX function is passed to a Netezza DBMS.

data work.newfunc;
   SASFUCNNAME = "SOUNDEX";
   SASFUCNNAMELEN = 7;
   DBMSFUCNNAME = "NYSIIS";
   DBMSFUCNNAMELEN = 6;
   FUNCTION_CATEGORY = "SCALAR";
   FUNC_USAGE_CONTEXT = "SELECT_LIST";
   FUNCTION_RETURNTYP = "CHAR";
   FUNCTION_NUM_ARGS = 1;
   CONVERT_ARGS = 0;
   ENGINEINDEX = 0;
output;
runk;

options sastrace=',,,d' sastraceloc=saslog nostsuffix;

/* Add function to existing in-memory function list */
libname x netezza server=&server. database=&database. uid=&user. pwd=&password.
   sql_functions="EXTERNAL_APPEND=work.newfunc" sql_functions_copy=saslog;

proc sql;
   select distinct soundex('oliver') from x._v_dual;
quit;

/* query returns */
/*
SQL_FUNCTIONS_COPY= LIBNAME Option

Writes the function associated with this particular LIBNAME statement to a SAS data set or the SAS log.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** none
- **Restrictions:** You must specify a two-part data set name, such as `<libref.member>` or an error results.
  `<libref.member>` must be a SAS data set. It is not checked to make sure that it is assigned to the default Base SAS engine.
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, MySQL, Netezza, ODBC, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** SQL_FUNCTIONS= LIBNAME option

**Syntax**

`SQL_FUNCTIONS_COPY=libref.member | SASLOG`

**Syntax Description**

- **`libref.member`**
  writes the current in-memory function list to a user-specified SAS data set for this particular LIBNAME statement.

- **SASLOG**
  writes the current in-memory function list to the SAS log for this particular LIBNAME statement.

SQL_OJ_ANSI= LIBNAME Option

Specifies whether to pass ANSI outer-join syntax through to the database.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** NO
- **Restriction:** Sybase can process SQL outer joins only if the version of the Adaptive Server Enterprise (ASE) database is 12.5.2 or higher.
- **Data source:** Sybase
Syntax

\texttt{SQL\_OJ\_ANSI=\text{YES} \mid \text{NO}}

\textbf{Syntax Description}

\textbf{YES}  
 specifies that ANSI outer-join syntax is passed through to the database.

\textbf{NO}  
 disables pass-through of ANSI outer-joins.

\textbf{SQLGENERATION= LIBNAME Option}

Specifies whether and when SAS procedures generate SQL for in-database processing of source data.

\begin{itemize}
  \item \textbf{Valid in:} SAS/ACCESS LIBNAME statement
  \item \textbf{Default:} none
  \item \textbf{Restrictions:} A value of NONE or DBMS modifies only the primary state that is set on the system option.
  \begin{itemize}
    \item If you are using the Metadata LIBNAME Engine, the only valid SQLGENERATION= modifiers are NONE and DBMS. The engine ignores the DBMS=, EXCLUDEDB=, and EXCLUDEPROC= modifiers.
  \end{itemize}
  \item \textbf{Data source:} Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Netezza, Oracle, SAP HANA, Teradata
  \item \textbf{Note:} Support for HAWQ was added in the third maintenance release for SAS 9.4.
  \item \textbf{See:} SQLGENERATION= system option, Precedence of Values for SQLGENERATION= LIBNAME and System Options, SAS In-Database Products: Administrator’s Guide
\end{itemize}

\textbf{Syntax}

\texttt{SQLGENERATION=\text{NONE} \mid \text{DBMS}}

\textbf{Required Arguments}

\textbf{NONE}  
 prevents those SAS procedures that are enabled for in-database processing from generating SQL for in-database processing.

\textbf{DBMS}  
 allows those SAS procedures that are enabled for in-database processing to generate SQL for in-database processing.

\textbf{STRINGDATES= LIBNAME Option}

Specifies whether to read date and time values from the database as character strings or as numeric date values.

\begin{itemize}
  \item \textbf{Valid in:} SAS/ACCESS LIBNAME statement, CONNECT statement
  \item \textbf{Default:} NO
\end{itemize}
**Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase IQ, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**Tip:** Use STRINGDATES=NO for SAS 6 compatibility.

### Syntax

```
STRINGDATES= YES | NO
```

### Syntax Description

**YES**

specifies that SAS reads date and time values as character strings.

**NO**

specifies that SAS reads date and time values as numeric date values.

### SYNONYMS= LIBNAME Option

Specifies whether PROC DATASETS shows synonyms, tables, views, or materialized views for the current user and schema if you specified the SCHEMA= option.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alias:</strong></td>
<td>SHOW_SYNONYMS=</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>YES</td>
</tr>
</tbody>
</table>

**Data source:** Netezza

**Note:** Support for this LIBNAME option was added for SAS 9.4.

**See:** SCHEMA= LIBNAME option, SCHEMA= data set option

### Syntax

```
SYNONYMS= YES | NO
```

### Syntax Description

**YES**

specifies that PROC DATASETS shows only synonyms that represent tables, views, or materialized views for the current user.

**NO**

specifies that PROC DATASETS shows only tables, views, or materialized views for the current user.

### Details

Rather than submit PROC DATASETS, you can select the libref in SAS Explorer to obtain this same information. By default, no PUBLIC synonyms are displayed unless you specify SCHEMA=PUBLIC.

When you specify only the SCHEMA option, the current schema is always displayed with the appropriate privileges.
INTERNALURATION

**TABLE_TYPE= LIBNAME Option**

Specifies the type of temporary tables and table storage when the engine creates tables in SAP HANA.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Access
- **Default:** none
- **Restrictions:**
  - If you do not specify a value for this option, tables that are created in SAP HANA follow the SAP HANA default for row or column store.
  - ROW and COLUMN are mutually exclusive.
  - LOCAL | LOCAL TEMPORARY and GLOBAL | GLOBAL TEMPORARY are mutually exclusive.
  - Do not specify LOCAL | LOCAL TEMPORARY or GLOBAL | GLOBAL TEMPORARY for permanent tables.
- **Interaction:** The TABLE_TYPE data set option overrides the TABLE_TYPE LIBNAME option.
- **Data source:** SAP HANA
- **See:** CONNECTION= (for sharing sessions across librefs), TABLE_TYPE= data set option
- **Examples:**
  - TABLE_TYPE=ROW
  - TABLE_TYPE=COLUMN
  - TABLE_TYPE=LOCAL
  - TABLE_TYPE=GLOBAL
  - TABLE_TYPE=(COLUMN GLOBAL)

**Syntax**

```
TABLE_TYPE=ROW | COLUMN
TABLE_TYPE=LOCAL | LOCAL TEMPORARY
TABLE_TYPE=GLOBAL | GLOBAL TEMPORARY
```

**Syntax Description**

- **ROW**
  - creates a table using ROW-based storage in SAP HANA.

- **COLUMN**
  - creates a table using COLUMN-based storage in SAP HANA.

- **LOCAL | LOCAL TEMPORARY**
  - creates a local temporary table in SAP HANA. The table definition and data are visible only in the current session.

- **GLOBAL | GLOBAL TEMPORARY**
  - creates a global temporary table in SAP HANA. The global temporary tables are globally available, and the data is visible only in the current session.
Details

This option takes affect when a SAS program creates a table in SAP HANA.

**TEMPORAL_QUALIFIER= LIBNAME Option**

Specifies time-dimension criteria for retrieving data from Teradata.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** CURRENT VALIDTIME [valid-time column], CURRENT TRANSACTIONTIME [transaction-time column]
- **Interaction:** Specifying values in a DATA step overrides LIBNAME values.
- **Data source:** Teradata
- **See:** TEMPORAL_QUALIFIER= data set option

**Syntax**

```
TEMPORAL_QUALIFIER= CURRENT VALIDTIME
| 'VALIDTIME AS OF PERIOD 'period' ' | SEQUENCED VALIDTIME
| 'NONSEQUENCED VALIDTIME 'period'

TEMPORAL_QUALIFIER= CURRENT TRANSACTIONTIME
| 'TRANSACTIONTIME AS OF PERIOD 'period'
| NONSEQUENCED TRANSACTIONTIME
```

**Syntax Description**

- **CURRENT VALIDTIME**
  selects rows that are valid at the current time.

- **VALIDTIME AS OF PERIOD 'period'**
  selects rows with valid-time periods that overlap the specified AS OF period. For the period, you can specify either a single date or a time period (date range) by specifying a start date and an end date.

- **SEQUENCED VALIDTIME PERIOD 'period'**
  selects history, current, or future rows that are valid for the specified time period.

- **NONSEQUENCED VALIDTIME PERIOD 'period'**
  treats the table as nontemporal.

- **CURRENT TRANSACTIONTIME**
  selects rows that are open in transaction time.

- **TRANSACTIONTIME AS OF 'period'**
  selects rows with transaction-time periods that overlap the specified AS OF period. For the period, you can specify either a single date or a time period (date range) by specifying a start date and an end date.

- **NONSEQUENCED TRANSACTIONTIME PERIOD 'period'**
  treats the table as nontemporal.

**Details**

Use temporal qualifiers to specify time criteria for selecting data from temporal tables.
Temporal qualifiers that you specify in a LIBNAME statement apply only to that Teradata session and are implemented through session commands that are issued at connect time. For example, if you specify TEMPORAL_QUALIFIER='AS OF PERIOD (1999-01-01, 2099-01-05)' in a LIBNAME statement, below is the Teradata SET SESSION command that is issued at connect time. The SQL is submitted as usual.

```
.SET SESSION ASOF PERIOD '(1999-01-01, 2099-01-05)'
```

Example: Select Valid-Time Rows for a Specific Date for the Current Session

In this example, valid-time rows are selected for a specific date from the mytest data set.

```
/* Consider data as of 1995-01-01. */
libname x teradata user=myusr1 pw=mypwd1 server=mysrv1
   TEMPORAL_QUALIFIER='VALIDTIME AS OF DATE '1995-01-01' ';
/* .SET SESSION VALIDTIME ASOF DATE'1995-01-01'; is issued
   before submitting the SQL "Select * from mytest" */
proc print data=x.mytest(DBSLICEPARM=ALL);
run;
```

TENACITY= LIBNAME Option

Specifies how many hours that FastExport, FastLoad, or MultiLoad continues to try to log on again to Teradata if the maximum number of Teradata utilities are already running.

Valid in: SAS/ACCESS LIBNAME statement

Default: 0 [FastLoad], 4 [FastExport, MultiLoad]

Data source: Teradata

Tip: The data set option has precedence over the LIBNAME option.

See: DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option, DBSLICEPARM= system option, FASTEXPORT= LIBNAME option, Maximizing Teradata Load Performance, MULTILOAD= data set option, SLEEP= LIBNAME option, SLEEP= data set option, TENACITY= data set option, Using the TPT API

Syntax

```
TENACITY=number-of-hours
```

Syntax Description

`number-of-hours`

specifies the number of hours to continue to try again to log on to Teradata.

Details

Use this option to indicate to FastExport, FastLoad, or MultiLoad how long to continue retrying a logon operation when the maximum number of utilities are already running. (The maximum number of Teradata utilities that can run concurrently varies from 5 to 15, depending on the database server setting.) The default value for TENACITY= is 4 hours. The value specified for TENACITY= must be greater than zero.
Use **TENACITY=** with **SLEEP=**. **SLEEP=** specifies the number of minutes that FastExport, FastLoad, or MultiLoad waits before it retries logging on to Terdata. **SLEEP=** and **TENACITY=** function very much like the **SLEEP** and **TENACITY** runtime options of the native Teradata FastExport, FastLoad, or MultiLoad utility.

Here is an example of the message that is written to the SAS log if the time period that **TENACITY=** specifies is exceeded.

```
ERROR: MultiLoad failed unexpectedly with returncode 12
```

Check the FastExport, FastLoad, or MultiLoad log for more information about the cause of the FastExport, FastLoad, or MultiLoad failure. SAS does not receive any informational messages from Teradata in either of these situations:

- when the currently run FastExport, FastLoad, or MultiLoad process waits because the maximum number of utilities are already running
- if FastExport, FastLoad, or MultiLoad is terminated because the time limit that **TENACITY=** specifies has been exceeded

The native Teradata FastExport, FastLoad, or MultiLoad utility sends messages associated with **SLEEP=** and **TENACITY=** only to the FastExport, FastLoad, or MultiLoad log. Therefore, nothing is written to the SAS log.

---

**TPT= LIBNAME Option**

Specifies whether SAS uses the Teradata Parallel Transporter (TPT) API to load data when SAS requests a Fastload, MultiLoad, or Multi-Statement insert.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>YES</td>
</tr>
<tr>
<td>Data source:</td>
<td>Teradata</td>
</tr>
<tr>
<td>See:</td>
<td>BULKLOAD= LIBNAME option, BULKLOAD= data set option, LOGDB= LIBNAME option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= data set option, TPT_APPL_PHASE= data set option, TPT_BUFFER_SIZE= data set option, TPT_CHECKPOINT= data set option, TPT_DATA_ENCRYPTION= data set option, TPT_ERROR_TABLE_1= data set option, TPT_ERROR_TABLE_2= data set option, TPT_LOG_TABLE= data set option, TPT_MAX_SESSIONS= LIBNAME option, TPT_MAX_SESSIONS= data set option, TPT_MIN_SESSIONS= data set option, TPT_PACK= data set option, TPT_PACKMAXIMUM= data set option, TPT_RESTART= data set option, TPT_TRACE_LEVEL= data set option, TPT_TRACE_LEVEL_INF= data set option, TPT_TRACE_OUTPUT= data set option, TPT_WORK_TABLE= data set option, &quot;Maximizing Teradata Load and Read Performance&quot;, &quot;Using the TPT API&quot;</td>
</tr>
</tbody>
</table>

**Syntax**

```
TPT=YES | NO
```

**Syntax Description**

**YES**

specifies that SAS uses the TPT API when Fastload, MultiLoad, or Multi-Statement insert is requested.
NO

specifies that SAS does not use the TPT API when Fastload, MultiLoad, or Multi-
Statement insert is requested.

Details

By using the TPT API, you can load data into a Teradata table without working directly
with such stand-alone Teradata utilities as FastLoad, MultiLoad, or TPump. When
TPT=NO, SAS uses the TPT API load driver for Fastload, the update driver for
MultiLoad, and the stream driver for Multi-Statement insert.

When TPT=YES, sometimes SAS cannot use the TPT API due to an error or because it
is not installed on the system. When this happens, SAS does not produce an error, but it
still tries to load data using the requested load method (Fastload, MultiLoad, or Multi-
Statement insert). To check whether SAS used the TPT API to load data, look for a
similar message to this one in the SAS log.

    NOTE: Teradata connection: TPT FastLoad/MultiLoad/MultiStatement
         insert has read n row(s).

Example: Load SAS Data into Teradata

In this example, SAS data is loaded into Teradata using the TPT API. This is the default
method of loading when Fastload, MultiLoad, or Multi-Statement insert are requested.
SAS still tries to load data even if it cannot use the TPT API.

    libname tera teradata user=myusr1 pw=mypwd1 TPT=YES;
    /* Create data */
    data testdata;
    do i=1 to 100;
        output;
    end;
    run;

    /* Load using MultiLoad TPT. This note appears in the SAS
     log if SAS uses TPT. NOTE: Teradata connection:
     TPT MultiLoad has inserted 100 row(s).*/
    data tera.testdata(MULTILOAD=YES);
    set testdata;
    run;

TPT_MAX_SESSIONS= LIBNAME Option

Specifies the maximum number of sessions for Teradata to use when using the TPT API with FastLoad,
MultiLoad, or Multi-Statement insert.

Valid in:   SAS/ACCESS LIBNAME statement
Default:   4
Restriction:  This option is valid only when using the TPT API.
Requirement: To use this option, you must first set TPT=YES.
Interaction: [precedence] 1. TPT_MAX_SESSIONS= data set option, 2. TPT_MAX_SESSIONS=
             LIBNAME option, 3. SAS_TPT_MAX_SESSIONS environment variable, 4. the
default value
**TPT_MAX_SESSIONS= LIBNAME Option**

Specifies the maximum number of sessions for Teradata to use when using the TPT API to load data with FastLoad, MultiLoad, or Multi-Statement insert.

**Syntax**

\[
\text{TPT\_MAX\_SESSIONS=integer}
\]

**Syntax Description**

*integer*

specifies the maximum number of sessions for Teradata to use when using the TPT API to load data with FastLoad, MultiLoad, or Multi-Statement insert.

**Details**

You can control the number of sessions for Teradata to use when using the TPT API to load data with MultiLoad. The maximum value cannot be more than the number of available Access Module Processors (AMPs). See your Teradata documentation for details.

**TPT_MIN_SESSIONS= LIBNAME Option**

Specifies the minimum number of sessions for Teradata to use when using the TPT API with FastLoad, MultiLoad, or Multi-Statement insert.

**Syntax**

\[
\text{TPT\_MIN\_SESSIONS=integer}
\]

**Detailed Description**

*integer*

specifies the minimum number of sessions for Teradata to use when using the TPT API to load data with FastLoad, MultiLoad, or Multi-Statement insert.

**Details**

You can control the number of sessions for Teradata to use when using the TPT API to load data with MultiLoad. The maximum value cannot be more than the number of available Access Module Processors (AMPs). See your Teradata documentation for details.
**Syntax Description**

integer

specifies the minimum number of sessions for Teradata to use when using the TPT API to load data with FastLoad, MultiLoad, or Multi-Statement insert.

**Details**

You can control the number of sessions for Teradata to use when using the TPT API to load data with MultiLoad. The minimum value cannot be more than the value for the TPT_MAX_SESSIONS= LIBNAME option.

---

**TR_ENABLE_INTERRUPT= LIBNAME Option**

Allows interruption of any long-running SQL processes that are involved in creating the result set or spool file.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Default:** NO
- **Restriction:** Valid for only Windows and UNIX platforms
- **Data source:** Teradata

**Syntax**

TR_ENABLE_INTERRUPT= YES | NO

**Syntax Description**

**YES**

allows interruption of long-running SQL processes that are involved in creating the result set or spool file.

**NO**

disables the interrupt processing code path.

**Details**

When set to YES, here is how you can use this option:

- to interrupt any SELECT SQL statement that was submitted by using the SELECT * FROM CONNECTION as a pass-through statement
- by using the Interrupt/Attention button to interrupt a query that you submitted through a DATA STEP, through PROC SQL implicit pass-through, or through explicit pass-through

Once the result set or spool file forms on the Teradata server and SAS is fetching the results, it is likely that the interrupt might no longer be available. You must wait until all results are fetched. The interrupt works in only one of these cases:

- when the Teradata server is building the result set or spool file
- if the Teradata server is in a wait state before building the result set or spool file because of locking
Example

```sas
libname x teradata user=myusr1 pass=mypwd1
   TR_ENABLE_INTERRUPT=YES server=mysrv1;

data _NULL_; set x.paul_test; run;

proc datasets lib=x; quit;

proc sql;
create table work.a as select * from x.td_cancel, x.td_cancel;
quit;

proc sql; connect to teradata (user=myusr1 pass=mypwd1
   TR_ENABLE_INTERRUPT=YES server=mysrv1);
select * from connection to teradata
   ( select * From td_cancel a , td_cancel b );
quit;
```

**TRACE= LIBNAME Option**

Specifies whether to turn on tracing information for use in debugging.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Default:** NO
- **Restriction:** This option is not supported on UNIX platforms.
- **Data source:** Aster, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, PostgreSQL, SAP HANA, Sybase IQ, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** TRACEFILE= LIBNAME option

**Syntax**

```
TRACE=YES | NO
```

**Syntax Description**

- **YES**
  - specifies that tracing is turned on, and the DBMS driver manager writes each function call to the trace file that TRACEFILE= specifies.

- **NO**
  - specifies that tracing is not turned on.

**TRACEFILE= LIBNAME Option**

Specifies the filename to which the DBMS driver manager writes trace information.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Default:** none
Restrictions: This option is not supported on UNIX platforms.
TRACEFILE= is used only when TRACE=YES.

Data source: Aster, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, PostgreSQL, SAP HANA, Sybase IQ, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: TRACE= LIBNAME option

Syntax

TRACEFILE=filename | '<path-and-filename>'

Details

If you specify a filename without a path, the SAS trace file is stored with your data files.
If you specify a directory, enclose the fully qualified filename in single quotation marks.
If you do not specify the TRACEFILE= option, output is directed to a default file.

TRANSCODE_FAIL= LIBNAME Option

Lets you specify how to handle processing and notification of transcoding errors.

Valid in: SAS/ACCESS LIBNAME statement
Default: ERROR
Restriction: This option is not available for use with SAS Embedded Process.
Data source: Hadoop
Tip: You can use TRANSCODE_FAIL= to determine whether you want to halt or continue processing when transcoding errors are encountered.

See: TRANSCODE_FAIL= data set option

Syntax

TRANSCODE_FAIL=<ERROR> | <WARNING> | <SILENT>

Optional Arguments

ERROR
stops processing data and provides an informative error message.

WARNING
continues processing of data but provides an informative error message.

SILENT
continues processing of data but suppresses messages.

UPDATE_ISOLATION_LEVEL= LIBNAME Option

Defines the degree of isolation of the current application process from other concurrently running application processes.
UPDATE_LOCK_TYPE= LIBNAME Option

Specifies how data in a DBMS table is locked during an update transaction.

Valid in: SAS/ACCESS LIBNAME statement
Default: none [DB2 under z/OS, Teradata], set by the data provider [OLE DB], NOLOCK [Oracle, Sybase], ROW [DB2 under UNIX and PC Hosts, Microsoft SQL Server, ODBC, PostgreSQL, Sybase IQ, Vertica]
Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, OLE DB, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica
Tip: You can set a lock for one DBMS table by using the data set option or for a group of DBMS tables by using the LIBNAME option.
See: CONNECTION= LIBNAME option, READ_LOCK_TYPE= LIBNAME option, READ_LOCK_TYPE= data set option, UPDATE_ISOLATION_LEVEL= LIBNAME option, UPDATE_LOCK_TYPE= data set option, UPDATE_LOCK_TYPE= data set option, SCHEMA= LIBNAME option, and DBMS-specific locking information in the reference section for your SAS/ACCESS interface
Syntax

UPDATE_LOCK_TYPE=ROW | PAGE | TABLE | NOLOCK | VIEW

Syntax Description

ROW [valid for DB2 under UNIX and PC Hosts, Microsoft SQL Server, ODBC, Oracle, PostgreSQL, Vertica]
locks a row if any of its columns are to be updated.

PAGE [valid for Sybase]
locks a page of data, which is a DBMS-specific number of bytes. This value is not valid for the Sybase interface when you use the .

TABLE [valid for DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, Oracle, Sybase IQ, Teradata]
locks the entire DBMS table.

NOLOCK [valid for Microsoft SQL Server, ODBC with Microsoft SQL Server driver, OLE DB, Oracle, Sybase]
does not lock the DBMS table, page, or any rows when reading them for update.

VIEW [valid for Teradata]
locks the entire DBMS view.

UPDATE_MODE_WAIT= LIBNAME Option

Specifies during SAS/ACCESS Update operations whether Teradata should wait to acquire a lock or fail the request when a different user has locked the DBMS resource.

Valid in: SAS/ACCESS LIBNAME statement
Default: none
Data source: Teradata
See: Locking in the Teradata Interface, UPDATE_MODE_WAIT= data set option

Syntax

UPDATE_MODE_WAIT=YES | NO

Syntax Description

YES
specifies for Teradata to wait to acquire the lock, so SAS/ACCESS waits indefinitely until it can acquire the lock.

NO
specifies that Teradata fails the lock request if the specified DBMS resource is locked.

Details

If you specify UPDATE_MODE_WAIT=NO and a different user holds a restrictive lock, the executing SAS step fails. SAS/ACCESS continues processing the job by executing the next step.
A restrictive lock means that a different user is holding a lock that prevents you from obtaining the lock that you want. Until the other user releases the restrictive lock, you cannot obtain your lock. For example, another user's table-level WRITE lock prevents you from obtaining a READ lock on the table.

Use SAS/ACCESS locking options only when the standard Teradata standard locking is undesirable.

**UPDATE_MULT_ROWS= LIBNAME Option**

Indicates whether SAS updates multiple rows from a data source, such as a DBMS table.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>NO</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, Greenplum, HAWQ, Microsoft SQL Server, Netezza, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase IQ, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>DELETE_MULT_ROWS= LIBNAME option</td>
</tr>
</tbody>
</table>

### Syntax

**UPDATE_MULT_ROWS=**YES | NO

### Syntax Description

**YES**

specifies that SAS/ACCESS processing continues if multiple rows are updated. This might produce unexpected results.

**NO**

specifies that SAS/ACCESS processing does not continue if multiple rows are updated.

### Details

Some providers do not handle this DBMS SQL statement well and therefore update more than the current row with this statement:

UPDATE...WHERE CURRENT OF CURSOR

**UPDATE_MULT_ROWS=** allows SAS/ACCESS to continue if multiple rows were updated.

**UPDATE_SQL= LIBNAME Option**

Determines the method that is used to update and delete rows in a data source.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>SAS/ACCESS LIBNAME statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaults:</td>
<td>YES [except for the Oracle drivers from Microsoft and Oracle]</td>
</tr>
<tr>
<td></td>
<td>NO [Oracle drivers from Microsoft and Oracle, which do not support Current-of-Cursor operations]</td>
</tr>
<tr>
<td>Data source:</td>
<td>Greenplum, HAWQ, Microsoft SQL Server, ODBC, Vertica</td>
</tr>
</tbody>
</table>
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: INSERT_SQL= LIBNAME option, UPDATE_SQL= data set option

Syntax

UPDATE_SQL=\texttt{YES} | \texttt{NO}

Syntax Description

\texttt{YES}

specifies that SAS/ACCESS uses Current-of-Cursor SQL to update or delete rows in a table.

\texttt{NO}

specifies that SAS/ACCESS uses the SQLSetPos() application programming interface (API) to update or delete rows in a table.

Details

This is the equivalent of update or delete for the INSERT_SQL= LIBNAME option. The default for Oracle drivers from Microsoft and Oracle is \texttt{NO} because these drivers do not support Current-Of-Cursor operations.

\textbf{UPDATEBUFF= LIBNAME Option}

Specifies the number of rows that are processed in a single DBMS Update or Delete operation.

\begin{itemize}
  \item \textbf{Valid in:} SAS/ACCESS LIBNAME statement
  \item \textbf{Default:} 1
  \item \textbf{Data source:} Oracle
  \item \textbf{See:} UPDATEBUFF= data set option
\end{itemize}

Syntax

UPDATEBUFF=\texttt{positive-integer}

Syntax Description

\texttt{positive-integer}

the number of rows in an operation. SAS allows the maximum that the DBMS allows.

Details

When updating with the VIEWTABLE window or the FSVIEW procedure, use UPDATEBUFF=1 to prevent the DBMS interface from trying to update multiple rows. By default, these features update only one observation at a time. They do this because they use record-level locking by default and therefore lock only the observation that is currently being edited.
USE_DATADIRECT= LIBNAME Option

Specifies the driver to use.

Valid in: SAS/ACCESS LIBNAME statement
Alias: USE_PROGRESS=
Default: NO
Data source: Impala

Syntax
USE_DATADIRECT=YES | NO

Syntax Description
YES
specifies to use the DataDirect driver.

NO
specifies to use the Cloudera driver.

USE_ODBC_CL= LIBNAME Option

Indicates whether the Driver Manager uses the ODBC Cursor Library.

Valid in: SAS/ACCESS LIBNAME statement and some DBMS-specific connection options. See the DBMS-specific reference section for details.
Default: NO
Data source: Aster, Microsoft SQL Server, Netezza, ODBC, PostgreSQL, SAP HANA
See: For more information about the ODBC Cursor Library, see your vendor-specific documentation.

Syntax
USE_ODBC_CL=YES | NO

Syntax Description
YES
specifies that the Driver Manager uses the ODBC Cursor Library. The ODBC Cursor Library supports block scrollable cursors and positioned update and delete statements.

NO
specifies that the Driver Manager uses the scrolling capabilities of the driver.
**UTILCONN_TRANSIENT= LIBNAME Option**

Enables utility connections to maintain or drop, as needed.

**Valid in:** SAS/ACCESS LIBNAME statement and some DBMS-specific connection options. See the DBMS-specific reference section for details.

**Defaults:**
- YES (DB2 under z/OS)
- NO (Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLEDB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica)

**Restriction:** UTILCONN_TRANSIENT= has no effect on engines that do not support utility connections.

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** DELETE_MULT_ROWS= LIBNAME option

---

**Syntax**

`UTILCONN_TRANSIENT=YES | NO`

**Syntax Description**

**NO**
- Specifies that a utility connection is maintained for the lifetime of the libref.

**YES**
- Specifies that a utility connection is automatically dropped as soon as it is no longer in use.

**Details**

A utility connection is used for engines that can lock system resources as a result of such operations as DELETE or RENAME. Queries on system tables or table indexes can also lock system resources. This connection prevents COMMIT statements that are issued to unlock system resources from being submitted on the same connection that is being used for table processing. Keeping COMMIT statements from table processing connection alleviates such problems that they can cause as invalidating cursors and committing pending updates on the tables that are being processed.

Because a utility connection exists for each LIBNAME statement, the number of connections to a DBMS can be large as multiple librefs are assigned across multiple SAS sessions. Setting UTILCONN_TRANSIENT=YES keeps these connections from existing when they are not being used. This setting reduces the number of current connections to the DBMS at any given point in time.

---

**WARN_BIGINT= LIBNAME Option**

Issues a warning in the SAS log if the BIGINT data type is in a DB2 table.
Valid in: SAS/ACCESS LIBNAME statement
Alias: BIGINT_WARN=
Default: NO
Data source: DB2 under UNIX and PC Hosts

Syntax
WARN_BIGINT= YES | NO

Syntax Description
YES
issues a warning in the SAS log if a BIGINT data type is in a DB2 table.

NO
specifies that the warning for a BIGINT data type does not appear in the SAS log.

Details
If a BIGINT data type is detected in the result set and if WARN_BIGINT=YES, the log contains this warning:

A column of type BIGINT was detected in the result set. As BIGINT values are stored in SAS as DOUBLE PRECISION values, you may receive inexact results if the BIGINT value has a precision greater than 15 digits. Consider using the DBSASTYPE option to convert the BIGINT column into character value to preserve the precision of the BIGINT value.
Chapter 12
Data Set Options for Relational Databases

About the Data Set Options for Relational Databases ........................................ 241

Overview .................................................................................................................. 241

Dictionary ................................................................................................................. 242

AUTHID= Data Set Option ....................................................................................... 242
AUTOCONNECT= Data Set Option ............................................................................. 242
BL_ALLOW_READ_ACCESS= Data Set Option ........................................................... 243
BL.Allow_WRITE_ACCESS= Data Set Option ............................................................. 243
BL_API_BULKLOAD= Data Set Option .................................................................. 244
BL.BADFILE= Data Set Option ............................................................................... 245
BL_CLIENT_DATAFILE= Data Set Option ............................................................... 246
BL.CODEPAGE= Data Set Option .......................................................................... 246
BL_CONTROL_FIELD_DELIMITER= Data Set Option ............................................. 248
BL_CONTROL_QUOTATION_MARK= Data Set Option ............................................... 248
BL_CONTROL= Data Set Option ............................................................................. 249
BL_COPY_LOCATION= Data Set Option .................................................................. 251
BL_CPU_PARALLELISM= Data Set Option .............................................................. 251
BL_DATA_BUFFER_SIZE= Data Set Option .............................................................. 252
BL_DATAFILE_EXISTS= Data Set Option ............................................................... 253
BL_DATAFILE_PATH= Data Set Option ................................................................... 253
BL_DATAFILE= Data Set Option ............................................................................ 254
BL_DATAFILE= Data Set Option [Teradata] ............................................................ 255
BL_DB2CURSOR= Data Set Option ........................................................................... 257
BL_DB2DATAACLAS= Data Set Option ................................................................... 258
BL_DB2DEVT_PERM= Data Set Option .................................................................. 258
BL_DB2DEVT_TEMP= Data Set Option .................................................................... 259
BL_DB2DISC= Data Set Option ............................................................................... 259
BL_DB2ERR= Data Set Option ................................................................................. 259
BL_DB2IN= Data Set Option ................................................................................... 260
BL_DB2LDCT1= Data Set Option ............................................................................ 261
BL_DB2LDCT2= Data Set Option ............................................................................ 261
BL_DB2LDCT3= Data Set Option ............................................................................ 262
BL_DB2LDEXT= Data Set Option ............................................................................ 262
BL_DB2MGMTCLAS= Data Set Option ..................................................................... 263
BL_DB2MAP= Data Set Option ................................................................................ 264
BL_DB2PRINT= Data Set Option ............................................................................. 264
BL_DB2PRNLOG= Data Set Option .......................................................................... 265
BL_DB2REC= Data Set Option .................................................................................. 265
BL_DB2RECS= Data Set Option ................................................................................ 266
BL_DB2RSTRT= Data Set Option ............................................................................. 266
BL_DB2SPC_PERM= Data Set Option ....................................................................... 267
<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_DB2SPC_TEMP</td>
<td>Data Set Option</td>
<td>267</td>
</tr>
<tr>
<td>BL_DB2STORCLAS</td>
<td>Data Set Option</td>
<td>268</td>
</tr>
<tr>
<td>BL_DB2TBLXST</td>
<td>Data Set Option</td>
<td>269</td>
</tr>
<tr>
<td>BL_DB2UNITCOUNT</td>
<td>Data Set Option</td>
<td>270</td>
</tr>
<tr>
<td>BL_DB2UTID</td>
<td>Data Set Option</td>
<td>270</td>
</tr>
<tr>
<td>BL_DBNAME</td>
<td>Data Set Option</td>
<td>271</td>
</tr>
<tr>
<td>BL_DEFAULT_DIR</td>
<td>Data Set Option</td>
<td>271</td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE</td>
<td>Data Set Option</td>
<td>272</td>
</tr>
<tr>
<td>BL_DELETE_ONLY_DATAFILE</td>
<td>Data Set Option</td>
<td>274</td>
</tr>
<tr>
<td>BL_DELIMITER</td>
<td>Data Set Option</td>
<td>275</td>
</tr>
<tr>
<td>BL_DIRECT_PATH</td>
<td>Data Set Option</td>
<td>277</td>
</tr>
<tr>
<td>BL_DISCARDFILE</td>
<td>Data Set Option</td>
<td>278</td>
</tr>
<tr>
<td>BL_DISK_PARALLELISM</td>
<td>Data Set Option</td>
<td>279</td>
</tr>
<tr>
<td>BL_ENCODING</td>
<td>Data Set Option</td>
<td>279</td>
</tr>
<tr>
<td>BL_ESCAPE</td>
<td>Data Set Option</td>
<td>280</td>
</tr>
<tr>
<td>BL_EXCEPTION</td>
<td>Data Set Option</td>
<td>281</td>
</tr>
<tr>
<td>BL_EXECUTE_CMD</td>
<td>Data Set Option</td>
<td>282</td>
</tr>
<tr>
<td>BL_EXECUTE_LOCATION</td>
<td>Data Set Option</td>
<td>282</td>
</tr>
<tr>
<td>BL_EXTERNAL_WEB</td>
<td>Data Set Option</td>
<td>283</td>
</tr>
<tr>
<td>BL_FILE_BADFILE</td>
<td>Data Set Option</td>
<td>284</td>
</tr>
<tr>
<td>BL_FILE_CONTROLFILE</td>
<td>Data Set Option</td>
<td>285</td>
</tr>
<tr>
<td>BL_FILE_DEFAULT_DIR</td>
<td>Data Set Option</td>
<td>286</td>
</tr>
<tr>
<td>BL_FILE_DELETE_DATAFILE</td>
<td>Data Set Option</td>
<td>286</td>
</tr>
<tr>
<td>BL_FORCE_NOT_NULL</td>
<td>Data Set Option</td>
<td>287</td>
</tr>
<tr>
<td>BL_FORMAT</td>
<td>Data Set Option</td>
<td>287</td>
</tr>
<tr>
<td>BL_HEADER</td>
<td>Data Set Option</td>
<td>288</td>
</tr>
<tr>
<td>BL_HOST</td>
<td>Data Set Option</td>
<td>289</td>
</tr>
<tr>
<td>BL_IMPORT_BATCH_SIZE</td>
<td>Data Set Option</td>
<td>289</td>
</tr>
<tr>
<td>BL_IMPORT_DIR</td>
<td>Data Set Option</td>
<td>289</td>
</tr>
<tr>
<td>BL_IMPORT_TABLE_LOCK</td>
<td>Data Set Option</td>
<td>290</td>
</tr>
<tr>
<td>BL_IMPORT_TYPE_CHECK</td>
<td>Data Set Option</td>
<td>291</td>
</tr>
<tr>
<td>BL_INDEX_OPTIONS</td>
<td>Data Set Option</td>
<td>291</td>
</tr>
<tr>
<td>BL_INDEXING_MODE</td>
<td>Data Set Option</td>
<td>292</td>
</tr>
<tr>
<td>BL_KEEPIDENTITY</td>
<td>Data Set Option</td>
<td>293</td>
</tr>
<tr>
<td>BL_KEEPNULLS</td>
<td>Data Set Option</td>
<td>294</td>
</tr>
<tr>
<td>BL_LOAD_METHOD</td>
<td>Data Set Option</td>
<td>294</td>
</tr>
<tr>
<td>BL_LOAD_REPLACE</td>
<td>Data Set Option</td>
<td>295</td>
</tr>
<tr>
<td>BL_LOCATION</td>
<td>Data Set Option</td>
<td>296</td>
</tr>
<tr>
<td>BL_LOG</td>
<td>Data Set Option</td>
<td>296</td>
</tr>
<tr>
<td>BL_LOGFILE</td>
<td>Data Set Option</td>
<td>297</td>
</tr>
<tr>
<td>BL_MAPFILE</td>
<td>Data Set Option</td>
<td>298</td>
</tr>
<tr>
<td>BL_METHOD</td>
<td>Data Set Option</td>
<td>299</td>
</tr>
<tr>
<td>BL_NULL</td>
<td>Data Set Option</td>
<td>299</td>
</tr>
<tr>
<td>BL_OPTIONS</td>
<td>Data Set Option</td>
<td>300</td>
</tr>
<tr>
<td>BL_PARFILE</td>
<td>Data Set Option</td>
<td>301</td>
</tr>
<tr>
<td>BL_PATH</td>
<td>Data Set Option</td>
<td>302</td>
</tr>
<tr>
<td>BL_PORT</td>
<td>Data Set Option</td>
<td>303</td>
</tr>
<tr>
<td>BL_PORT_MAX</td>
<td>Data Set Option</td>
<td>304</td>
</tr>
<tr>
<td>BL_PORT_MIN</td>
<td>Data Set Option</td>
<td>304</td>
</tr>
<tr>
<td>BL_PREERVECLUDED</td>
<td>Data Set Option</td>
<td>305</td>
</tr>
<tr>
<td>BL_PROTOCOL</td>
<td>Data Set Option</td>
<td>305</td>
</tr>
<tr>
<td>BL_PSQL_PATH</td>
<td>Data Set Option</td>
<td>306</td>
</tr>
<tr>
<td>BL_QUOTE</td>
<td>Data Set Option</td>
<td>306</td>
</tr>
<tr>
<td>BL_RECOVERABLE</td>
<td>Data Set Option</td>
<td>307</td>
</tr>
<tr>
<td>BL_REJECT_LIMIT</td>
<td>Data Set Option</td>
<td>308</td>
</tr>
</tbody>
</table>
BL_REJECT_TYPE= Data Set Option .................................................. 308
BL.REMOTE_FILE= Data Set Option .................................................. 309
BL_RETURN_WARNINGS_AS_ERRORS= Data Set Option ......................... 310
BL_SERVER_DATAFILE= Data Set Option .......................................... 310
BL_SFTP_HOST= Data Set Option ...................................................... 311
BL_SFTP_OPTIONS= Data Set Option ............................................... 312
BL_SFTP_USER= Data Set Option .................................................... 312
BL_SFTP_WAIT_MILLISECONDS= Data Set Option ............................... 313
BL_SQLDR_PATH= Data Set Option .................................................... 313
BL_SUPPRESS_NULLIF= Data Set Option ........................................... 314
BL_USE_PIPE= Data Set Option ....................................................... 315
BL.WARNING_COUNT= Data Set Option .............................................. 316
BUFFERS= Data Set Option .......................................................... 316
BULK_BUFFER= Data Set Option ....................................................... 317
BULKLOAD= Data Set Option ........................................................ 317
BULKUNLOAD= Data Set Option ....................................................... 318
BUSINESS_DATATYPE= Data Set Option ........................................... 319
BUSINESS_TIMEFRAME= Data Set Option ......................................... 319
CAST= Data Set Option .............................................................. 320
CAST_OVERHEAD_MAXPERCENT= Data Set Option .............................. 321
CHAR_AS_BINARY= Data Set Option ................................................ 322
COLUMN_DELIMITER= Data Set Option ............................................. 322
COMMAND_TIMEOUT= Data Set Option ............................................. 323
CONFIG= Data Set Option ........................................................... 323
CONFIGDIR= Data Set Option ........................................................ 324
CURSOR_TYPE= Data Set Option ..................................................... 324
DATETIME2= Data Set Option ........................................................ 326
DB.ONE_CONNECT.PER_THREAD= Data Set Option ............................. 326
DBCOMMIT= Data Set Option ......................................................... 327
DBCONDITION= Data Set Option ..................................................... 328
DBCONSTRAINT= Data Set Option ................................................... 329
DBCREATE_TABLE_EXTERNAL= Data Set Option ............................... 330
DBCREATE_TABLE_LOCATION= Data Set Option ................................ 331
DBCREATE_TABLE_OPTS= Data Set Option ........................................ 332
DBFORCE= Data Set Option .......................................................... 333
DBGEN_NAME= Data Set Option ...................................................... 334
DBINDEX= Data Set Option .......................................................... 335
DBKEY= Data Set Option ............................................................ 337
DBLABEL= Data Set Option .......................................................... 338
DBLINK= Data Set Option ............................................................ 339
DBMASTER= Data Set Option ........................................................ 340
DBMAX_TEXT= Data Set Option ....................................................... 341
DBNULL= Data Set Option ............................................................ 342
DBNULLKEYS= Data Set Option ....................................................... 343
DBPROMPT= Data Set Option ........................................................ 344
DBSASLABEL= Data Set Option ....................................................... 345
DBSASTYPE= Data Set Option ....................................................... 346
DBSLICE= Data Set Option ........................................................... 347
DBSLICEPARAM= Data Set Option .................................................... 349
DBTYPE= Data Set Option ............................................................ 351
DEGREE= Data Set Option ............................................................ 353
DIMENSION= Data Set Option ....................................................... 354
DISTRIBUTE_ON= Data Set Option ................................................ 354
DISTRIBUTED_BY= Data Set Option ............................................... 355
ERRLIMIT= Data Set Option .......................................................... 356
ESCAPE_BACKSLASH= Data Set Option ........................................... 357
Chapter 12 • Data Set Options for Relational Databases

- FASTEXP= Data Set Option .......................................................... 358
- FASTLOAD= Data Set Option .......................................................... 360
- FETCH_IDENTITY= Data Set Option ................................................. 361
- HDFS_PRINCIPAL= Data Set Option ............................................... 362
- IGNORE_READ_ONLY_COLUMNS= Data Set Option ............................... 362
- IN= Data Set Option ................................................................. 364
- INSERT_SQL= Data Set Option ........................................................ 364
- INSERTBUFF= Data Set Option ........................................................ 365
- KEYSET_SIZE= Data Set Option ..................................................... 366
- LOCATION= Data Set Option ......................................................... 367
- LOCKTABLE= Data Set Option ....................................................... 367
- MBUF= Data Set Option ............................................................... 368
- ML_CHECKPOINT= Data Set Option ................................................ 368
- ML_ERROR1= Data Set Option ........................................................ 369
- ML_ERROR2= Data Set Option ........................................................ 370
- ML_LOG= Data Set Option ............................................................ 371
- ML_RESTART= Data Set Option ...................................................... 372
- ML_WORK= Data Set Option .......................................................... 373
- MULTLOAD= Data Set Option ........................................................ 373
- MULTISTMT= Data Set Option ........................................................ 379
- NULLCHAR= Data Set Option ........................................................ 381
- NULLCHARVAL= Data Set Option ................................................... 382
- OR_IDENTITY_COLS= Data Set Option ............................................ 382
- OR_PARTITION= Data Set Option .................................................... 383
- OR_UPD_NOWHERE= Data Set Option ............................................... 386
- ORHINTS= Data Set Option .......................................................... 387
- OVERLAPS= Data Set Option ......................................................... 388
- PARMDEFAULT= Data Set Option .................................................... 389
- PARMSTRING= Data Set Option ...................................................... 389
- PARTITION_KEY= Data Set Option ................................................ 390
- PARTITIONED_BY= Data Set Option ................................................ 391
- POST_STMT_OPTS= Data Set Option ................................................. 392
- POST_TABLE_OPTS= Data Set Option .............................................. 393
- PRE_STMT_OPTS= Data Set Option .................................................. 394
- PRE_TABLE_OPTS= Data Set Option ............................................... 395
- PRESERVE_COL_NAMES= Data Set Option ....................................... 397
- QUALIFIER= Data Set Option ....................................................... 398
- QUERY_BAND= Data Set Option ..................................................... 399
- QUERY_TIMEOUT= Data Set Option ................................................ 399
- READBUFF= Data Set Option ......................................................... 400
- READ_ISOLATION_LEVEL= Data Set Option ...................................... 401
- READ_LOCK_TYPE= Data Set Option ............................................... 401
- READ_METHOD= Data Set Option ................................................... 402
- READ_MODE_WAIT= Data Set Option ................................................ 403
- ROW_DELIMITER= Data Set Option ................................................. 404
- SASDATEFMT= Data Set Option ..................................................... 404
- SCHEMA= Data Set Option ............................................................ 406
- SEGMENT_NAME= Data Set Option .................................................. 406
- SESSIONS= Data Set Option ......................................................... 408
- SET= Data Set Option ................................................................. 408
- SLEEP= Data Set Option .............................................................. 409
- SYSTEM_TIMEFRAME= Data Set Option ......................................... 410
- TABLE_TYPE= Data Set Option ...................................................... 411
- TEMPORAL= Data Set Option ....................................................... 412
- TEMPORAL_QUALIFIER= Data Set Option ........................................ 412
- TENACITY= Data Set Option ........................................................ 415
About the Data Set Options for Relational Databases

Overview

You can specify SAS/ACCESS data set options on a SAS data set when you access DBMS data with the SAS/ACCESS LIBNAME statement. A data set option applies only to the data set on which it is specified, and it remains in effect for the duration of the DATA step or procedure. For options that you can assign to a group of relational DBMS tables or views, see “LIBNAME Options for Relational Databases” on page 101.

Here is an example of how you can SAS/ACCESS data set options.

```
libname myoralib oracle;
proc print myoralib.mytable(data-set-option=value)
```

You can also use SAS/ACCESS data set options on a SAS data set when you access DBMS data using access descriptors. See “Using Descriptors with the ACCESS Procedure” on page 970. Here is an example.

```
proc print mylib.myviewd(data-set-option=value)
```

You cannot use most data set options on a PROC SQL DROP (table or view) statement. You can use the CNTLLEV=, DROP=, FIRSTOBS=, IN=, KEEP=, OBS=, RENAME=, and WHERE= SAS data set options when you access DBMS data. SAS/ACCESS
interfaces do not support the REPLACE= SAS data set option. For information about using SAS data set options, see *SAS Data Set Options: Reference*.

The information in this section explains all applicable data set options. The information includes DBMS support and the corresponding LIBNAME options, and refers you to documentation for your SAS/ACCESS interface when appropriate. For a list of the data set options available in your SAS/ACCESS interface with default values, see the DBMS-specific reference section for your SAS/ACCESS interface.

Specifying data set options in PROC SQL might reduce performance, because it prevents operations from being passed to the DBMS for processing. For more information, see “Overview: Optimizing Your SQL Usage” on page 43.

---

**Dictionary**

---

**AUTHID= Data Set Option**

Lets you qualify the specified table with an authorization ID, user ID, or group ID.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>SCHEMA=</td>
</tr>
<tr>
<td>Default</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>Data source</td>
<td>DB2 under z/OS</td>
</tr>
<tr>
<td>See</td>
<td>AUTHID= LIBNAME option</td>
</tr>
</tbody>
</table>

**Syntax**

AUTHID=authorization-ID

**Syntax Description**

*authorization-ID*

limited to eight characters.

**Details**

If you specify a value for the AUTHID= option, the table name is qualified as authid.tablename before any SQL code is passed to the DBMS. If you do not specify AUTHID=, the table name is not qualified before being passed to the DBMS and the DBMS uses your user ID as the qualifier. If you specify AUTHID= in a SAS/SHARE LIBNAME statement, the ID of the active server is the default ID.

---

**AUTOCOMMIT= Data Set Option**

Specifies whether to enable the DBMS autocommit capability.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>Data source</td>
<td>MySQL, Sybase</td>
</tr>
</tbody>
</table>
See: AUTOCOMMIT= LIBNAME option, DBCOMMIT= data set option

Syntax
AUTOCOMMIT= YES | NO

Syntax Description
YES
specifies that SAS commits all updates, inserts, and deletes immediately after they
are executed and that no rollback is possible.

NO
specifies that SAS commits after processing the number of rows that DBCOMMIT=
specifies. If you do not specify DBCOMMIT=, SAS commits after processing the
default number of rows.

BL_ALLOW_READ_ACCESS= Data Set Option
Specifies that the original table data is still visible to readers during bulk loading.
Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Requirement: To specify this option, you must first set BULKLOAD= YES.
Data source: DB2 under UNIX and PC Hosts
See: BL_ALLOW_WRITE_ACCESS= data set option, BULKLOAD= data set option

Syntax
BL_ALLOW_READ_ACCESS= YES | NO

Syntax Description
YES
specifies that the original (unchanged) data in the table is still visible to readers while
bulk loading is in progress.

NO
specifies that readers cannot view the original data in the table while bulk loading is
in progress.

Details
For more information about using this option, see the
SQLU.Allow.Read.Access parameter in the IBM DB2 Universal Database Data
Movement Utilities Guide and Reference.

BL_ALLOW_WRITE_ACCESS= Data Set Option
Specifies that table data is still accessible to readers and writers while import is in progress.
Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: DB2 under UNIX and PC Hosts
See: BL_ALLOW_READ_ACCESS= data set option, BULKLOAD= data set option

Syntax
BL_ALLOW_WRITE_ACCESS=YES | NO

Syntax Description
YES
specifies that table data is still visible to readers and writers during data import.

NO
specifies that readers and writers cannot view table data during data import.

Details
For more information about using this option, see the SQLU_ALLOW_WRITE_ACCESS parameter in the IBM DB2 Universal Database Data Movement Utilities Guide and Reference.

BL_API_BULKLOAD= Data Set Option
Specifies to perform bulk loading using the Oracle Direct Path API instead of the Oracle SQL*Loader utility.
Valid in: DATA and PROC steps (when accessing data using SAS/ACCESS software)
Default: NO
Restriction: This data set option is not valid in z/OS operating environments.
Requirement: To specify this option, you must first set BULKLOAD=YES
Data source: Oracle
Note: Support for this data set option was added in the second maintenance release for SAS 9.4.
See: BULKLOAD= data set option, ERRLIMIT= data set option, INSERTBUFF= data set option

Syntax
BL_API_BULKLOAD=YES | NO

Syntax Description
YES
specifies to perform bulk loading using the Direct Path API.

Notes
The Oracle SQL* Loader is not required for bulk loading using Direct Path API.
Direct Path API bulk loading shows better performance for SAS tables that contain mainly numeric values as compared to the same bulk loading using the SQL*Loader utility.

NO
specifies to perform bulk loading using the Oracle SQL*Loader utility.

Details
The following are differences in bulk loading when you use the Direct Path API:

- DAT files are not needed. Therefore, they are not created.
- The SQL*Loader utility is not used.
- Other bulk-loading data set options are ignored for Direct Path API bulk loading.
- Direct Path API bulk loading uses the value of the INSERTBUFF= data set option to optimize performance. It uses the ERLIMIT= data set option to limit the number of errors before SAS stops processing and issues a rollback.

**BL_BADFILE= Data Set Option**
Identifies a file that contains records that were rejected during bulk loading.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software) |
| Default: | creates a data file in the current directory or with the default file specifications |
| Requirement: | To specify this option, you must first set BULKLOAD=YES. |
| Data source: | Oracle |
| See: | BULKLOAD= data set option |

**Syntax**

```
BL_BADFILE=path-and-filename
```

**Syntax Description**

`path-and-filename`

an SQL*Loader file to which rejected rows of data are written. On most platforms, the default filename takes the form BL_\(<table>\_<unique-ID>\).bad:

- `table`
specifies the table name
- `unique-ID`
specifies a number that is used to prevent collisions in the event of two or more simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates the number.

**Details**
If you do not specify this option and a BAD file does not exist, a file is created in the current directory (or with the default file specifications). If you do not specify this option
and a BAD file already exists, the Oracle bulk loader reuses the file, replacing the contents with rejected rows from the new load.

Either Oracle or the SQL*Loader can reject records. For example, the SQL*Loader can reject a record that contains invalid input, and Oracle can reject a record because it does not contain a unique key. If no records are rejected, the BAD file is not created.

On most operating systems, the BAD file is created in the same format as the DATA file, so the rejected records can be loaded after corrections have been made.

Operating Environment Information
On z/OS operating systems, the BAD file is created with default DCB attributes. For details about overriding this, see the information about SQL*Loader file attributes in the SQL*Loader chapter in your Oracle user's guide for z/OS.

---

**BL_CLIENT_DATAFILE= Data Set Option**

Specifies the client view of the data file that contains DBMS data for bulk loading.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** the current directory

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** Sybase IQ

**See:** BULKLOAD= data set option

**Syntax**

```
BL_CLIENT_DATAFILE=path-and-data-filename
```

**Syntax Description**

**path-and-data-filename**

specifies the file that contains the rows of data to load or append into a DBMS table during bulk loading. On most platforms, the default filename takes the form

```
BL_<table>_<unique-ID>.dat
```

- **table**
  - specifies the table name.

- **unique-ID**
  - specifies a number that is used to prevent collisions in the event of two or more simultaneous bulk-loading operations on a particular table. The SAS/ACCESS engine generates the number.

- **dat**
  - specifies the .dat file extension for the data file.

---

**BL_CODEPAGE= Data Set Option**

Identifies the code page that the DBMS engine uses to convert SAS character data to the current database code page during bulk loading.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: the code page ID of the current window

Requirements: The value for this option must never be 0. If you do not wish any code page conversions to take place, use the BL_OPTIONS= option to specify 'FORCEIN'. Code page conversions occur for only DB2 character data types.

To specify this option, you must first set BULKLOAD=YES.

Data source: DB2 under UNIX and PC Hosts

See: BL_OPTIONS= data set option, BULKLOAD= data set option

Syntax

BL_CODEPAGE=numeric-code-page-ID

Syntax Description

numeric-code-page-ID

a numeric value that represents a character set that is used to interpret multibyte character data and determine the character values.

BL_CONTROL_FIELD_DELIMITER= Data Set Option

Specifies override of the default delimiter character for separating columns of data during data transfer or retrieval during bulk loading.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Access

Aliases: BL_CONTROL_FIELD_DELIMITER

BL_DELIMITER

BL_FIELD_DELIMITER

Default: . (comma)

Range: You can specify any single-character.

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: SAP HANA

Note: When you specify this option, FIELD DELIMITED BY field-delimiter is added to the SAP HANA IMPORT statement.

See: BL_CONTROL_QUOTATION_MARK= data set option,

BL_CONTROL_RECORD_DELIMITER= data set option, BL_FILE_BADFILE= data set option, BL_FILE_CONTROLFILE= data set option, BL_FILE_DATAFILE= data set option, BL_FILE_DEFAULT_DIR= data set option,

BL_FILE_DELETE_DATAFILE= data set option, BULKLOAD= data set option

Syntax

BL_CONTROL_FIELD_DELIMITER='<field-delimiter>'
**BL_CONTROL_QUOTATION_MARK= Data Set Option**

Specifies the quotation character for CSV mode.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Access
- **Aliases:** BL_CONTROL_QUOTATION_MARK
  - BL_QUOTATION
  - BLQUOTE
- **Default:** " (double quote)
- **Requirement:** To specify this option, you must first set BULKLOAD=Yes.
- **Data source:** SAP HANA

See: BL_CONTROL_FIELD_DELIMITER= data set option, BL_CONTROL_RECORD_DELIMITER= data set option, BL_FILE_BADFILE= data set option, BL_FILE_CONTROLFILE= data set option, BL_FILE_DATAFILE= data set option, BL_FILE_DEFAULT_DIR= data set option, BL_FILE_DELETE_DATAFILE= data set option, BULKLOAD= data set option

**Syntax**

\[ \text{BL\_CONTROL\_QUOTATION\_MARK=quotation-mark} \]

---

**BL_CONTROL_RECORD_DELIMITER= Data Set Option**

Specifies the single delimiter character to use to separate SAP HANA table records.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Access
- **Aliases:** BL_CONTROL_RECORD_DELIMITER
  - RECORD_DELIMITER
- **Default:** \n (new line)
- **Range:** You can specify any single-character.
- **Requirement:** To specify this option, you must first set BULKLOAD=Yes.
- **Data source:** SAP HANA

**Note:** When you specify this option, RECORD DELIMITED BY record-delimiter is added to the SAP HANA IMPORT statement.

See: BL_CONTROL_FIELD_DELIMITER= data set option, BL_CONTROL_QUOTATION_MARK= data set option, BL_FILE_BADFILE= data set option, BL_FILE_CONTROLFILE= data set option, BL_FILE_DATAFILE= data set option, BL_FILE_DEFAULT_DIR= data set option, BL_FILE_DELETE_DATAFILE= data set option, BULKLOAD= data set option
BL_CONTROL= Data Set Option

Identifies the file that contains control statements.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Alias: FE_EXECNAME [Teradata]
Default: DBMS-specific
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: Oracle, Teradata
See: BL_DATAFILE= data set option, BL_DELETE_DATAFILE= data set option,
BL_DELETE_ONLY_DATAFILE= data set option, BULKLOAD= data set option,
DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option

Syntax

BL_CONTROL=Record_DELIMITER='<record-delimiter>'

Syntax Description

path-and-control-filename [Oracle]
specifies the SQL*Loader file (where SQLLDR control statements are written) that
describe the data to include in bulk loading.

path-and-data-filename [Teradata]
specifies the name of the control file to generate for extracting data with
SAS/ACCESS Interface to Teradata using FastExport multithreaded Read.

BL_<table>_<_unique-ID>.ctl [Oracle, Teradata (UNIX or PC Hosts)]
the default filename on most platforms, where:

table
specifies the table name

unique-ID
specifies a number that is used to prevent collisions in the event of two or more
simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates
the number.

Details

Specifics for Oracle
The Oracle interface creates the control file by using information from the input data and
SAS/ACCESS options. The file contains Data Definition Language (DDL) definitions
that specify the location of the data and how the data corresponds to the database table. It
is used to specify exactly how the loader should interpret the data that you are loading
from the DATA file (DAT file). By default it creates a control file in the current directory
or with the default file specifications. If you do not specify this option and a control file
does not already exist, a file is created in the current directory or with the default file specifications. If you do not specify this option and a control file already exists, the Oracle interface reuses the file and replaces the contents with the new control statements.

**Specifics for Teradata**

To specify this option, you must first set DBSCLICEPARM=ALL as a LIBNAME or data set option for threaded Reads. By default SAS creates a data file in the current directory or with a platform-specific name. If you do not specify this option and a control file does not exist, SAS creates a script file in the current directory or with the default file specifications. If you do not specify this option and a control file already exists, the DATA step. SAS/ACCESS Interface to Teradata creates the control file by using information from the input data and SAS/ACCESS options. The file contains FastExport Language definitions that specify the location of the data and how the data corresponds to the database table. It is used to specify exactly how the FastExport should interpret the data that you are loading from the DATA (.DAT) file. Because the script file that SAS generates for FastExport must contain login information in clear text, it is recommended that you secure the script file by specifying a directory path that is protected.

**File Naming for Teradata under z/OS**

By default under z/OS, the control filename is prefixed with your user ID. This happens automatically without having to specify it. However, you can suppress the addition of your user ID to the filename and add text of your choice instead. Typically, you would add a security group ID to the beginning of the filename in place of your user ID. When the addition of your user ID is suppressed, you specify the text to add to the beginning of the filename. To prevent adding your user ID to the filename, enable the TD_RACF environment variable when you start SAS. To set the TD_RACF environment variable, include the following code in your SAS command:

```plaintext
-SET "TD_RACF YES"
```

Alternatively, you can specify TD_RACF as a SAS option.

*Note:* You can use the TD_RACF environment variable even when you use a security method other than RACF.

For more information, see “Example 2: Under z/OS: Add a Security Group Name to the Filename” on page 257.

**Example: Generate Teradata Script Files**

This example generates a Teradata script file, C:\protodir\fe.ctl on Windows.

```plaintext
DATA test;
SET teralib.mydata(DBSCLICEPARM=ALL BL_CONTROL="C:\protodir\fe.ctl");
run;
```

This example generates a Teradata script file, /tmp/fe.ctl, on UNIX.

```plaintext
DATA test;
SET teralib.mydata(DBSCLICEPARM=ALL BL_CONTROL="/tmp/fe.ctl");
run;
```

This example generates a script file, USERID.SECURE.SCR.CTL, by appending CTL and prepending the user ID.

```plaintext
DATA test;
SET teralib.mydata(DBSCLICEPARM=ALL BL_CONTROL="SECURE.SCR");
```
**BL_COPY_LOCATION= Data Set Option**

Specifies the directory to which DB2 saves a copy of the loaded data.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES and BL_RECOVERABLE=YES.
- **Data source:** DB2 under UNIX and PC Hosts
- **See:** BL_RECOVERABLE= data set option, BULKLOAD= data set option

**Syntax**

`BL_COPY_LOCATION=pathname`

**BL_CPU_PARALLELISM= Data Set Option**

Specifies the number of processes or threads to use when building table objects.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under UNIX and PC Hosts
- **See:** BL_DATA_BUFFER_SIZE= data set option, BL_DISK_PARALLELISM= data set option, BULKLOAD= data set option

**Syntax**

`BL_CPU_PARALLELISM=number of processes or threads`

**Syntax Description**

`number of processes or threads`

specifies the number of processes or threads that the load utility uses to parse, convert, and format data records when building table objects.

**Details**

This option exploits intrapartition parallelism and significantly improves load performance. It is particularly useful when loading presorted data, because record order in the source data is preserved.

The maximum allowable number is 30. If the value is 0 or has not been specified, the load utility selects an intelligent default. This default is based on the number of available CPUs on the system at run time. If there is insufficient memory to support the specified value, the utility adjusts the value.
When BL_CPU_PARALLELISM is greater than 1, the flushing operations are asynchronous, permitting the loader to exploit the CPU. If tables include either LOB or LONG VARCHAR data, parallelism is not supported. The value is set to 1 regardless of the number of system CPUs or the specified value.

Although use of this parameter is not restricted to symmetric multiprocessor (SMP) hardware, you might not obtain any discernible performance benefit from using it in non-SMP environments.

For more information about using BL_CPU_PARALLELISM=, see the CPU_PARALLELISM parameter in the *IBM DB2 Universal Database Data Movement Utilities Guide and Reference*.

---

**BL_DATA_BUFFER_SIZE= Data Set Option**

Specifies the total amount of memory to allocate for the bulk-load utility to use as a buffer for transferring data.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under UNIX and PC Hosts
- **See:** BL_CPU_PARALLELISM= data set option, BL_DISK_PARALLELISM= data set option, BULKLOAD= data set option

**Syntax**

```
BL_DATA_BUFFER_SIZE=buffer-size
```

**Syntax Description**

- `buffer-size` specifies the total amount of memory that is allocated for the bulk-load utility to use as buffered space for transferring data within the utility. Memory is calculated in 4 KB pages regardless of the degree of parallelism.

**Details**

If you specify a value that is less than the algorithmic minimum, the minimum required resource is used and no warning is returned. This memory is allocated directly from the utility heap, the size of which you can modify through the `util_heap_sz` database configuration parameter. If you do not specify a value, the utility calculates an intelligent default at run time. This calculated default is based on a percentage of the free space that is available in the utility heap at the time of instantiation of the loader, as well as on some characteristics of the table.

It is recommended that the buffer be several extents in size. An *extent* is the unit of movement for data within DB2, and the extent size can be one or more 4KB pages. The DATA BUFFER parameter is useful when you are working with large objects (LOBs) because it reduces input and output waiting time. The data buffer is allocated from the utility heap. Depending on the amount of storage available on your system, you should consider allocating more memory for use by the DB2 utilities. You can modify the database configuration parameter `util_heap_sz` accordingly. The default value for the Utility Heap Size configuration parameter is 5000 4KB pages. Because load is only one
of several utilities that use memory from the utility heap, it is recommended that no more than 50% of the pages defined by this parameter be made available for the load utility.

For more information about using this option, see the DATA BUFFER parameter in the IBM DB2 Universal Database Data Movement Utilities Guide and Reference.

---

**BL_DATAFILE_EXISTS= Data Set Option**

Lets you load a table from an existing data set.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: NO

Restriction: You cannot use this option if you set the BL_USE_PIPE= data set option.

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: Greenplum, HAWQ

Notes: Support for this data set option was added for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: BL_USE_PIPE= data set option, BULKLOAD= LIBNAME option on page 112, BULKLOAD= data set option

---

**Syntax**

BL_DATAFILE_EXISTS=YES | NO

**Optional Arguments**

YES

indicates that the engine opens the specified file in Read mode if the data set already exists.

NO

indicates that the engine creates the specified file and opens it in Load mode if the data set does not already exist.

---

**BL_DATAFILE_PATH= Data Set Option**

Specifies a path for creating a flat file for bulk loading.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: none

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: Aster

Note: Support for this data set option was added for SAS 9.4.

See: BULKLOAD= data set option
Syntax
DATAFILE_PATH=path

Syntax Description
path
specifies that SAS creates one or more temporary tables.

Details
When you use this option to set the path, it uses the automatically generated filename where the temporary flat file is created for bulk loading. If you do not specify this option, a file is created in the directory from which SAS runs.

Example: Set a Path for a Temporary File
libname dblib aster dsn=ncluster uid=user pwd=password dimension=yes;

data dblib.class (BULKLOAD=YES BL_DATAFILE_PATH='C:\temp\');
  set sashelp.class;
run;

BL_DATAFILE= Data Set Option
Identifies the file that contains DBMS data for bulk loading.

Valid in: DATA and PROC steps (when accessing data using SAS/ACCESS software)
Default: DBMS-specific
Requirement: To specify this option, you must first set BULKEXTRACT=YES or BULKLOAD=YES.
Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Netezza, Oracle, PostgreSQL, Sybase IQ
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: BL_DATAFILE= [Teradata], BL_CONTROL= data set option, BL_DELETE_DATAFILE= data set option, BL_DELETE_ONLY_DATAFILE= data set option, BL_PROTOCOL= data set option, BL_USE_PIPE= data set option, BULKLOAD= data set option

Syntax
BL_DATAFILE= path-and-data-filename

Syntax Description
path-and-data-filename
specifies the file that contains the rows of data to load or append into a DBMS table during bulk loading. On most platforms, the default filename takes the form
BL_<table>_<unique-ID>.ext:
table
specifies the table name.
unique-ID
specifies a number that is used to prevent collisions in the event of two or more simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates the number.

ext
specifies the file extension (.DAT or .IXF) for the data file.

Details

DB2 under UNIX and PC Hosts: The default is the current directory.

Greenplum, HAWQ: This option specifies the name of the external file to load. It is meaningful only when BL_PROTOCOL= is set to gpfdist or file. If you do not specify this option, the filename is generated automatically. When you specify the filename with a full path, the path overrides the value of the GLOAD_HOME environment variable. However, bulk loading might fail if the path does not match the base directory that the gpfdist utility used.

Netezza: You can use this option only when BL_USE_PIPE=NO. By default the SAS/ACCESS engine creates a data file from the input SAS data set in the current directory or with the default file specifications before calling the bulk loader. The data file contains SAS data that is ready to load into the DBMS. By default, the data file is deleted after the load is completed. To override this behavior, specify BL_DELETE_DATAFILE=NO.

Oracle: The SAS/ACCESS engine creates this data file from the input SAS data set before calling the bulk loader. The data file contains SAS data that is ready to load into the DBMS. By default, the data file is deleted after the load is completed. To override this behavior, specify BL_DELETE_DATAFILE=NO. If you do not specify this option and a data file does not exist, the file is created in the current directory or with the default file specifications. If you do not specify this option and a data file already exists, SAS/ACCESS reuses the file, replacing the contents with the new data. SAS/ACCESS Interface to Oracle on z/OS is the exception: The data file is never reused because the interface causes bulk loading to fail instead of reusing a data file.

Sybase IQ: By default, the SAS/ACCESS engine creates a data file with a .dat file extension in the current directory or with the default file specifications. Also, by default, the data file is deleted after the load is completed. To override this behavior, specify BL_DELETE_DATAFILE=NO.
Syntax

**BL_DATAFILE=** *path-and-data-filename*

**Syntax Description**

*path-and-data-filename*

specifies the name of the control file to generate for loading data with SAS/ACCESS Interface to Teradata using MultiLoad. On Windows and UNIX platforms, the default filename takes the form *BL_<table>_<unique-ID>.ctl*:

- **table**
  - specifies the table name.

- **unique-ID**
  - specifies a number that is used to prevent collisions in the event of two or more simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates the number.

On z/OS platforms, the default filename takes the form *<unique-ID>.<table>.dat*:

- **unique-ID**
  - specifies the user or group ID that owns the file. By default, this is your user ID. However, you can set this to a security group ID with the TD_RACF environment variable.

- **table**
  - specifies the table name.

**Details**

**Overview of BL_DATAFILE= for Teradata**

The file contains MultiLoad Language definitions that specify the location of the data and how the data corresponds to the database table. It specifies exactly how MultiLoad should interpret the data that you are loading. Because the script file that SAS generates for MultiLoad must contain login information in clear text, you should secure the script file by specifying a protected directory path.

**Specifics for File Naming under z/OS**

By default under z/OS, the control filename is prefixed with your user ID. This happens automatically without having to specify it. However, you can suppress the addition of your user ID to the filename and add text of your choice instead. Typically, you would add a security group ID to the beginning of the filename in place of your user ID. When the addition of your user ID is suppressed, you specify the text to add to the beginning of the filename. To prevent adding your user ID to the filename, enable the TD_RACF environment variable when you start SAS. To set the TD_RACF environment variable, include the following code in your SAS command:

```
-SET "TD_RACF YES"
```

Alternatively, you can specify TD_RACF as a SAS option.

**Note:** You can use the TD_RACF environment variable even when you use a security method other than RACF.
Examples

Example 1: Generate Teradata Script Files
This example generates a Teradata script file, C:\prot\dir\\ml.ctl, on Windows.

DATA teralib.test(DBSLICEPARM=ALL BL_DATAFILE="C:\prot\dir\\ml.ctl");
SET teralib.mydata;
run;

This next example generates a Teradata script file, fe.ctl, for FastExport and ml.ctl for MultiLoad.

data teralib.test1(MULTILOAD=YES TPT=NO BL_DATAFILE="ml.ctl");
SET teralib.test2(DBSLICEPARM=ALL BL_CONTROL="fe.ctl");
run;

Example 2: Under z/OS: Add a Security Group Name to the Filename
This example shows how to specify that generated filenames should include the dbi2 security group rather than the user ID.

/* the new env variable / SAS option TD_RACF should be set to YES when invoking
 S AS in order to test any scenario:
 * * 
 * sas -set TD_RACF YES -set SYSIN SASIN
 * */

libname tera teradata user=dbitest pw=XXXXX tdpid=tdp0;
proc delete data=tera.multitest ; run;

/* Multiload utility */
data tera.multitest(multiload=yes
   bi_datafile='dbi2.test.datafile'
   bi_control='dbi2.test.control');
set sashelp.class;
run;

BL_DB2CURSOR= Data Set Option
Specifies a string that contains a valid DB2 SELECT statement that points to either local or remote objects (tables or views).

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: DB2 under z/OS
See: BULKLOAD= data set option

Syntax
BL_DB2CURSOR=’SELECT * from filename’
Details

You can use it to load DB2 tables directly from other DB2 and objects that are not DB2. However, before you can select data from a remote location, your database administrator must first populate the communication database with the appropriate entries.

**BL_DB2DATACLAS= Data Set Option**

Specifies a data class for a new SMS-managed data set.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BL_DB2IN= data set option, BL_DB2MGMTCLAS= data set option, BL_DB2PRINT= data set option, BL_DB2STORCLAS= data set option [contains sample code], BL_DB2UNITCOUNT= data set option, BULKLOAD= data set option

**Syntax**

```
BL_DB2DATACLAS=data-class
```

**Details**

This option applies to the control file (BL_DB2IN= data set option), the input file (BL_DB2REC= data set option), and the output file (BL_DB2PRINT= data set option) for the bulk loader. Use this option to specify a data class for a new SMS-managed data set. SMS ignores this option if you specify it for a data set that SMS does not support. If SMS is not installed or active, the operating environment ignores any data class that BL_DB2DATACLAS= passes. Your site storage administrator defines the data class names that you can specify when you use this option.

**BL_DB2DEVT_PERM= Data Set Option**

Specifies the unit address or generic device type to use for permanent data sets that the LOAD utility creates—also SYSIN, SYSREC, and SYSPRINT when SAS allocates them.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** SYSDA
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_DB2DEVT_PERM=unit-specification
```
**BL_DB2DEVT_TEMP= Data Set Option**

Specifies the unit address or generic device type to use for temporary data sets that the LOAD utility creates (Pnch, Copy1, Copy2, RCpy1, RCpy2, Work1, Work2).

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** SYSDA

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** DB2 under z/OS

**See:** BULKLOAD= data set option

**Syntax**

BL_DB2DEVT_TEMP=unit-specification

---

**BL_DB2DISC= Data Set Option**

Specifies the SYSDISC data set name for the LOAD utility.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** a generated data set name

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** DB2 under z/OS

**See:** BULKLOAD= data set option, DB2PRINT= data set option

**Syntax**

BL_DB2DISC=data-set-name

**Details**

The DSNUTILS procedure with DISP=(NEW,CATLG,CATLG) allocates this option. This option must be the name of a nonexistent data set, except on a RESTART because it would already have been created. The LOAD utility allocates it as DISP=(MOD,CATLG,CATLG) on a RESTART. The default is a generated data set name, which appears in output that is written to the DB2PRINT= data set option location.

---

**BL_DB2ERR= Data Set Option**

Specifies the SYSERR data set name for the LOAD utility.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** a generated data set name

**Requirement:** To specify this option, you must first set BULKLOAD=YES.
Data source: DB2 under z/OS
See: BULKLOAD= data set option, DB2PRINT= data set option

Syntax

BL_DB2ERR=data-set-name

Details

The DSNUTILS procedure with DISP=(NEW,CATLG,CATLG) allocates this option. This option must be the name of a nonexistent data set, except on a RESTART because it would already have been created. The LOAD utility allocates it as DISP=(MOD,CATLG,CATLG) on a RESTART. The default is a generated data set name, which appears in output that is written to the DB2PRINT= data set option location.

BL_DB2IN= Data Set Option

Specifies the SYSIN data set name for the LOAD utility.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: a generated data set name
Requirement: To specify this option, you must first set BULKLOAD=YES.
Interaction: In SAS 9.4, the default value for LRECL is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.
Data source: DB2 under z/OS
See: BULKLOAD= data set option, BL_DB2LDEXT= data set option

Syntax

BL_DB2IN=data-set-name

Details

This option is allocated based on the value of BL_DB2LDEXT=. It is initially allocated as SPACE=(trk,(10,1),rlse) with the default being a generated data set name, which appears in the DB2PRINT output, with these DCB attributes:

DSORG=PS    LRECL=516
RECFM=VB    BLKSZE=23476

It supports these DCB attributes for existing data sets:

DSORG=PS
RECFM=F, FB, FS, FBS, V, VB, VS, or VBS
LRECL=any valid value for RECFM
BLKSIZE=any valid value for RECFM
**BL_DB2LDCT1= Data Set Option**

Specifies a string in the LOAD utility control statement between LOAD DATA and INTO TABLE.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>DB2 under z/OS</td>
</tr>
<tr>
<td>See:</td>
<td>BULKLOAD= data set option, BL_DB2LDCT2= data set option, BL_DB2DCT3= data set optionBL_DB2LDEXT= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

`BL_DB2LDCT1='string'`

**Details**

This option specifies a string that contains a segment of the Load Utility Control Statement between 'LOAD DATA' and 'INTO TABLE'. Valid control statement options include but are not limited to RESUME, REPLACE, LOG, and ENFORCE.

You can use DB2 bulk-load control options (BL_DB2LDCT1=, BL_DB2LDCT2=, and BL_DB2DCT3=) to specify sections of the control statement, which the engine incorporates into the control statement that it generates. These options have no effect when BL_DB2LDEXT=USERUN. You can use these options as an alternative to specifying BL_DB2LDEXT=GENONLY and editing the control statement to include options that the engine cannot generate. In some cases, it is necessary to specify at least one of these options. An example is running the utility on an existing table where you must specify either RESUME or REPLACE.

The LOAD utility requires that the control statement be in uppercase—except for objects such as table or column names, which must match the table. You must specify values for DB2 bulk-load control options using the correct case. SAS/ACCESS Interface to DB2 under z/OS cannot convert the entire control statement to uppercase because it might contain table or column names that must remain in lowercase.

**BL_DB2LDCT2= Data Set Option**

Specifies a string in the LOAD utility control statement between INTO TABLE `table-name` and `(field-specification)`.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>DB2 under z/OS</td>
</tr>
<tr>
<td>See:</td>
<td>BULKLOAD= data set option</td>
</tr>
</tbody>
</table>
**BL_DB2LDCT2= Data Set Option**

Specifies a string in the LOAD utility control statement after (field-specification).

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_DB2LDCT2='string'
```

**Details**

Valid control statement options include but are not limited to PART, PREFORMAT, RESUME, REPLACE, and WHEN.

---

**BL_DB2LDCT3= Data Set Option**

Specifies a string in the LOAD utility control statement after (field-specification).

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_DB2LDCT3='string'
```

**Details**

This option handles any options that might be defined for this location in later versions of DB2.

---

**BL_DB2LDEXT= Data Set Option**

Specifies the mode of execution for the DB2 LOAD utility.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** GENRUN
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Interaction:** In SAS 9.4, the default value for LRECL is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.
- **Data source:** DB2 under z/OS
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_DB2LDEXT=GENRUN | GENONLY | USERUN
```
Syntax Description

**GENRUN**

generates the control (SYSIN) file and the data (SYSREC) file, and runs the utility with them.

**GENONLY**

generates the control (SYSIN) file and the data (SYSREC) file but does not run the utility. Use this method when you need to edit the control file or to verify the generated control statement or data before you run the utility.

**USERUN**

uses existing control and data files, and runs the utility with them. Existing files can be from a previous run or from previously run batch utility jobs. Use this method when you restart a previously stopped run of the utility.

All valid data sets that the utility accepts are supported when `BL_DB2LDEXT=USERUN`. However, syntax errors from the utility can occur because no parsing is done when reading in the SYSIN data set. Specifically, neither embedded comments (beginning with a double hyphen, ‘--’) nor columns 73 through 80 of RECFM=FB LRECL=80 data sets are stripped from the control statement. The solution is to remove embedded comments and columns 73 through 80 of RECFM=FB LRECL=80 data sets from the data set. However, this is not an issue when you use engine-generated SYSIN data sets because they are RECFM=VB and therefore have no embedded comments.

Details

This option specifies the mode of execution for the DB2 LOAD utility, which involves creating data sets that the utility needs and to call the utility.

---

**BL_DB2MGMTCLAS= Data Set Option**

Specifies a management class for a new SMS-managed data set.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** `BL_DB2DATACLAS= data set option`, `BL_DB2REC=`, `BL_DB2PRINT=`, `BL_DB2STORCLAS= data set option [contains sample code]`, `BL_DB2UNITCOUNT= data set option`, `BULKLOAD= data set option`

---

**Syntax**

```
BL_DB2MGMTCLAS=management-class
```

---

**Details**

This option applies to the control file (BL_DB2IN=), the input file (BL_DB2REC=), and the output file (BL_DB2PRINT=) for the bulk loader. Use this option to specify a management class for a new SMS-managed data set. If SMS is not installed or active, the operating environment ignores any management class that `BL_DB2MGMTCLAS=`
passes. Your site storage administrator defines the management class names that you can specify when you use this option.

**BL_DB2MAP= Data Set Option**
Specifies the SYSMAP data set name for the LOAD utility.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** a generated data set name
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_DB2MAP=data-set-name
```

**Details**
The DSNUTILS procedure with DISP=(NEW,CATLG,CATLG) allocates this option. This option must be the name of a nonexistent data set, except on a RESTART because it would already have been created. The LOAD utility allocates it as DISP=(MOD,CATLG,CATLG) on a RESTART. The default is a generated data set name, which appears in output that is written to the DB2PRINT location.

**BL_DB2PRINT= Data Set Option**
Specifies the SYSPRINT data set name for the LOAD utility.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** a generated data set name
- **Requirement:** To specify this option, you must first set BULKLOAD=YES. You must also specify BL_DB2PRNLOG=YES so that you can see the generated data set name in the SAS log.
- **Interaction:** In SAS 9.4, the default value for LRECL is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.
- **Data source:** DB2 under z/OS
- **See:** BL_DB2PRNLOG= data set option [for generated data set name in the SAS log], BULKLOAD= data set option

**Syntax**

```
BL_DB2PRINT=data-set-name
```
Details
This option is allocated with DISP=(NEW,CATLG,DELETE) and SPACE=(trk, (10,1),rlse). The default is a generated data set name, which appears in the DB2PRINT DSN, with these DCB attributes:

- DSORG=PS
- LRECL=258
- RECFM=VBA
- BLKSIZE=262–32767

BL_DB2PRNLOG= Data Set Option
Determines whether to write SYSPRINT output to the SAS log.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** YES
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BULKLOAD= data set option, DB2 under z/OS Bulk Loading

Syntax

BL_DB2PRNLOG= YES | NO

Syntax Description

**YES**

specifies that SYSPRINT output is written to the SAS log.

**NO**

specifies that SYSPRINT output is not written to the SAS log.

BL_DB2REC= Data Set Option
Specifies the SYSREC data set name for the LOAD utility.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** a generated data set name
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Interaction:** In SAS 9.4, the default value for LRECL is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.
- **Data source:** DB2 under z/OS
- **See:** BL_DB2LDEXT= data set option, BL_DB2RECSP= data set option, BULKLOAD= data set option

Syntax

BL_DB2REC=data-set-name
Details

The value of BL_DB2LDEXT= allocates this option. It is initially allocated as
SPACE=(cyl,(BL_DB2RECSP, 10%(BL_DB2RECSP)),rlse). The default is a generated
data set name, which appears in output that is written to the DB2PRINT data set name. It
supports these DCB attributes for existing data sets:

| DSORG=PS | LRECL=any valid value for RECFM |
| RECFM=FB | BLKSIZE=any valid value for RECFM |

---

**BL_DB2RECSP= Data Set Option**

Determines the number of cylinders to specify as the primary allocation for the SYSREC data set when it is created.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software) |
| Default: | 10 |
| Requirement: | To specify this option, you must first set BULKLOAD=YES. |
| Data source: | DB2 under z/OS |
| See: | BULKLOAD= data set option |

---

**Syntax**

BL_DB2RECSP=primary-allocation

**Details**

The secondary allocation is 10% of the primary allocation.

---

**BL_DB2RSTRT= Data Set Option**

Tells the LOAD utility whether the current load is a restart and, if so, indicates where to begin.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software) |
| Default: | NO |
| Requirements: | To specify this option, you must first set BULKLOAD=YES. |
| When you specify a value other than NO for BL_DB2RSTRT=, you must also specify and BL_DB2LDEXT=USERUNBL_DB2TBLXST=YES. |
| Data source: | DB2 under z/OS |
| See: | BL_DB2TBLXST= data set option, BL_DB2LDEXT= data set option, BULKLOAD= data set option |

---

**Syntax**

BL_DB2RSTRT=NO | CURRENT | PHASE
Syntax Description

NO
specifies a new run (not restart) of the LOAD utility.

CURRENT
specifies to restart at the last commit point.

PHASE
specifies to restart at the beginning of the current phase.

**BL_DB2SPC_PERM= Data Set Option**

Determines the number of cylinders to specify as the primary allocation for permanent data sets that the LOAD utility creates.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>10</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>DB2 under z/OS</td>
</tr>
<tr>
<td>See:</td>
<td>BULKLOAD= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

**BL_DB2SPC_PERM=primary-allocation**

**Details**

Permanent data sets are Disc, Maps, and Err. The DSNUTILS procedure controls the secondary allocation, which is 10% of the primary allocation.

**BL_DB2SPC_TEMP= Data Set Option**

Determines the number of cylinders to specify as the primary allocation for temporary data sets that the LOAD utility creates.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>10</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>DB2 under z/OS</td>
</tr>
<tr>
<td>See:</td>
<td>BULKLOAD= data set option, DB2 under z/OS Bulk Loading</td>
</tr>
</tbody>
</table>

**Syntax**

**BL_DB2SPC_TEMP=primary-allocation**
BL_DB2STORCLAS= Data Set Option

Specifies a storage class for a new SMS-managed data set.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: DB2 under z/OS
See: BL_DB2DATACLAS= data set option, BL_DB2DEVT_PERM= data set option,
BL_DB2IN= data set option, BL_DB2MGMTCLAS= data set option, BL_DB2PRINT= data set option,
BL_DB2REC= data set option, BL_DB2UNITCOUNT= data set option

Syntax

BL_DB2STORCLAS=storage-class

Details

A storage class contains the attributes that identify a storage service level that SMS uses for storage of the data set. It replaces any storage attributes that you specify in BL_DB2DEVT_PERM=.

This option applies to the control file (BL_DB2IN), the input file (BL_DB2REC=), and the output file (BL_DB2PRINT=) for the bulk loader. Use this option to specify a management class for a new SMS-managed data set. If SMS is not installed or active, the operating environment ignores any storage class that BL_DB2MGMTCLAS= passes. Your site storage administrator defines the storage class names that you can specify when you use this option.

Example: Generate SMS-Managed Control and Data Files

This example generates SMS-managed control and data files. It does not create the table, and you do not need to run the utility to load it.

```
libname db2lib db2 ssid=db2a;
data db2lib.customers (bulkload=yes
        bl_db2ldext=genonly
        bl_db2in='myusr1.sysin'
        bl_db2rec=myusr1.sysrec'
        bl_db2tblxst=yes
        bl_db2ldct1='REPLACE'
        bl_db2dataclas='STD'
        bl_db2mgmtclas='STD'
        bl_db2storclas='STD');
set work.customers;
run;
```
BL_DB2TBLXST= Data Set Option

Indicates whether the LOAD utility runs against an existing table.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BL_DB2LDEXT= data set option, BL_DB2LDICT1= data set option, BL_DB2LDICT2 data set option

### Syntax

`BL_DB2TBLXST= YES | NO`

### Syntax Description

**YES**
- specifies that the LOAD utility runs against an existing table. This is *not* a replacement operation. (See “Details” on page 269.)

**NO**
- specifies that the LOAD utility does not run against an existing table.

### Details

SAS/ACCESS does not currently support table replacement. You cannot simply create a new copy of an existing table to replace the original table. Instead, you must delete the table and then create a new version of it.

The DB2 LOAD utility does not create tables—it loads data into existing tables. The DB2 under z/OS interface creates a table before loading data into it—whether you use SQL INSERT statements or start the LOAD utility.

You might want to start the utility for an existing table that the DB2 engine did not create. If so, specify `BL_DB2TBLXST=YES` to tell the engine that the table already exists. When `BL_DB2TBLXST=YES`, the engine neither verifies that the table does not already exist, which eliminates the NO REPLACE error, nor creates the table.

BULKLOAD= is not valid for update opening of tables, which includes appending to an existing table. Therefore, to accomplish appending, use either `BL_DB2TBLXST= with an output open (normally creates the table)` or the LOAD utility against a previously created table. You can also use `BL_DB2TBLXST= with BL_DB2LDEXT=GENONLY if the table does not yet exist and you do not want to create or load it yet`. In this case, the control and data files are generated but the table is neither created nor loaded.

Because the table might be empty or might contain rows, specify the appropriate LOAD utility control statement values for REPLACE, RESUME, or both by using `BL_DB2LDICT1`, `BL_DB2LDICT2`, or both.

The data to be loaded into the existing table must match the table column types. The engine does not try to verify input data with the table definition. The LOAC utility flags any incompatible differences.
**BL_DB2UNITCOUNT= Data Set Option**

Specifies the number of volumes on which data sets can be extended.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Restriction:** An error is returned if you specify a value for this option that exceeds the maximum number of volumes for the unit.
- **Requirements:** To specify this option, you must first set BULKLOAD=YES.

This option applies only to the input file (BL_DB2REC data set). This is the file that must be loaded into the DB2 table.

- **Data source:** DB2 under z/OS
- **See:** BL_DB2DATACLAS= data set option, BL_DB2DEVT_PERM= data set option, BL_DB2STORCLAS= data set option [contains sample code], BL_DB2STORCLAS= data set option, BULKLOAD= data set option

**Syntax**

```
BL_DB2UNITCOUNT=number-of-volumes
```

**Syntax Description**

- `number-of-volumes` specifies the number of volumes across which data sets can be extended. It must be an integer between 1 and 59. This option is ignored if the value is greater than 59. See the details in this section.

**Details**

You must specify an integer from 1–59 as a value for this option. This option is ignored if the value is greater than 59. However, the value depends on the unit name in BL_DB2DEVT_PERM=. At the operating environment level an association exists that defines the maximum number of volumes for a unit name. Ask your storage administrator for this number.

The data class determines whether SMS-managed data sets can be extended on multiple volumes. When you specify both BL_DB2DATACLAS= and BL_DB2UNITCOUNT=, BL_DB2UNITCOUNT= overrides the unit count values for the data class.

---

**BL_DB2UTID= Data Set Option**

Specifies a unique identifier for a given run of the DB2 LOAD utility.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** user ID and second level DSN qualifier
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under z/OS
- **See:** BULKLOAD= data set option
**Syntax**

\texttt{BL\_DB2UTID=utility-ID}

**Syntax Description**

\textit{utility-ID}

- a character string up to 16 bytes long.

**Details**

By default, the value for this option is the user ID concatenated with the second-level data set name qualifier. The generated ID appears in output that is written to the DB2PRINT data set name. This name generation makes it easy to associate all information for each utility execution and to separate it from other executions.

---

**BL\_DBNAME= Data Set Option**

Specifies the database name to use for bulk loading.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirements:** To specify this option, you must first set BULKLOAD=YES.
  - You must enclose the database name in quotation marks.
- **Data source:** Aster

**See:** \texttt{BL\_HOST= data set option, BL\_PATH= data set option, BULKLOAD= data set option}

**Syntax**

\texttt{BL\_DBNAME='database-name'}

**Syntax Description**

\textit{database-name}

- specifies the database name to use for bulk loading.

**Details**

Use this option to pass the database name to the DBMS bulk-load facility.

---

**BL\_DEFAULT\_DIR= Data Set Option**

Specifies where bulk loading creates all intermediate files.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Defaults:** \texttt{database-name} (Oracle)
  - \texttt{current-directory} (PostgreSQL)
- **Requirements:** To specify this option, you must first set BULKLOAD=YES.
This option must end with a backslash on Windows or forward slash on UNIX.

**Supports:** CTL, DAT, LOG, BAD, DSC intermediate bulk-load files (Oracle) or CTL and DAT intermediate bulk-load files (PostgreSQL)

**Data source:** Oracle, PostgreSQL

**See:** BULKLOAD= data set option

**Examples:**

```plaintext
BL_DEFAULT_DIR='c:\temp\'

[Windows]
BL_DEFAULT_DIR='/temp/'

[UNIX]
```

**Syntax**

```plaintext
BL_DEFAULT_DIR='host-specific-directory-path'
```

**Required Argument**

*host-specific-directory-path*

specifies the host-specific directory path where intermediate bulk-load files are created.

**Details**

The value that you specify for this option is prepended to the filename. Be sure to provide the complete, host-specific directory path, including the file and directory separator character to accommodate all platforms.

**Example: Create All Files in a Temporary Directory**

In this example, bulk loading creates all related files in the C:\temp directory.

```plaintext
data x.test (bulkload=yes BL_DEFAULT_DIR="c:\temp\" bl_delete_files=no);
c1=1;
run;
```

---

**BL_DELETE_DATAFILE= Data Set Option**

Specifies whether to delete only the data file or all files that the SAS/ACCESS engine creates for the DBMS bulk-load facility.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Alias:** BL_DELETE_FILES= [Oracle]

**Default:** YES

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Netezza, Oracle, PostgreSQL, Sybase IQ

**Notes:** Support for PostgreSQL was added for SAS 9.4. Support for Impala was added in the second maintenance release for SAS 9.4.
Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: BL_CONTROL= data set option, BL_DATAFILE= data set option, BL_DELETE_ONLY_DATAFILE= data set option, BL_USE_PIPE= data set option, BULKLOAD= data set option

Syntax

BL_DELETE_DATAFILE=YES | NO

Syntax Description

YES
deletes all (data, control, and log) files that the SAS/ACCESS engine creates for the DBMS bulk-load facility.

NO
does not delete these files.

Details

DB2 under UNIX and PC Hosts: Setting BL_DELETE_DATAFILE=YES deletes only the temporary data file that SAS/ACCESS creates after the load completes.

Greenplum, HAWQ: When BL_DELETE_DATAFILE=YES, the external data file is deleted after the load completes.

Netezza: You can use this option only when BL_USE_PIPE=NO.

Oracle, PostgreSQL: When BL_DELETE_DATAFILE=YES, all files (DAT, CTL, and LOG) are deleted.

Examples

Example 1: Delete All Files
In this example the default is YES, so all files are deleted.

libname x oracle &connopts
proc datasets library=x;
   delete test1;run;
data x.test1 ( bulkload=yes );
c1=1;
run;
x dir BL_TEST1*.*;

Example 2: Retain All Files
No files are deleted in this example.

libname x oracle &connopts
proc datasets library=x;
   delete test2;run;
data x.test2 ( bulkload=yes bl_delete_files=no );
c1=1;
run;
x dir BL_TEST2*.*;
**BL_DELETE_ONLY_DATAFILE= Data Set Option**

Specifies whether to delete the data file that the SAS/ACCESS engine creates for the DBMS bulk-load facility.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Oracle

**See:** BL_CONTROL= data set option, BL_DATAFILE= data set option, BL_DELETE_DATAFILE= data set option

**Syntax**

```
BL_DELETE_ONLY_DATAFILE=YES | NO
```

**Syntax Description**

- **YES**
  - deletes only the data file that the SAS/ACCESS engine creates for the DBMS bulk-load facility.
- **NO**
  - does not delete the data file.

**Details**

Setting this option overrides the BL_DELETE_DATAFILE= option.

**Examples**

**Example 1: Delete Only the Control and Log Files**

BL_DELETE_DATAFILE=YES is the default in this example, so only the control and log files are deleted.

```sas
proc datasets library=x;
  delete test3;run;
data x.test3 ( bulkload=yes bl_delete_only_datafile=no );
c1=1;
run;
x dir BL_TEST3*.*;
```

**Example 2: Retain All Files**

Both options are set to NO in this example, so no files are deleted.

```sas
proc datasets library=x;
  delete test4;run;
data x.test4 ( bulkload=yes bl_delete_only_datafile=no bl_delete_files=NO );
c1=1;
run;
x dir BL_TEST4*.*;
```
**Example 3: Delete Only the Data File**

Only the data file is deleted in this example.

```plaintext
proc datasets library=x;
   delete test5;run;
data x.test5 ( bulkload=yes bl_delete_only_datafile=YES );
c1=1;
run;
x dir BL_TEST5*.*;
```

The same is true in this example.

```plaintext
proc datasets library=x;
   delete test6;run;
run;
data x.test6 ( bulkload=yes bl_delete_only_datafile=YES bl_delete_files=NO );
c1=1;
run;
x dir BL_TEST6*.*;
```

**BL_DELIMITER= Data Set Option**

Specifies override of the default delimiter character for separating columns of data during data transfer or retrieval during bulk loading or unloading.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Alias:** DELIM, DELIMIT [Hadoop]
- **Default:** DBMS-specific
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Aster, Greenplum, Hadoop, HAWQ, Netezza, PostgreSQL
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** BL_DATAFILE= data set option, BL_DELETE_DATAFILE= data set option, BL_FORCE_NOT_NULL= data set option, BL_FORMAT= data set option, BL_NULL= data set option, BL_OPTIONS= data set option, BLQUOTE= data set option, BLUSEPIPE= data set option, BULKLOAD= data set option, BULKUNLOAD= LIBNAME option, BULKUNLOAD= data set option

**Syntax**

```plaintext
BL_DELIMITER='<any-single-character>'
```

**Details**

**Overview**

Here is when you might want to use this option:

- to override the default delimiter character that the interface uses to separate columns of data that are transferred to or retrieved from the DBMS during bulk loading. For Netezza, this also applies to bulk unloading.
- if your character data contains the default delimiter character, to avoid any problems while parsing the data stream
You must ensure that the characters that are assigned to BL_DELIMITER= and BLQUOTE= are different.

**Aster Specifics**
The default is `/t` (the tab character).

**Greenplum Specifics**
The default is the pipe symbol (`|`).

**Hadoop Specifics**
The default is `\001` (Ctrl-A). To change the default delimiter, specify a value as either a single character or three-digit decimal ASCII value between 001 and 127. The value represents the ASCII value of the delimiter that you want to use. You cannot use other typical SAS or UNIX formats such as ‘\001’, 0x01 or ‘01’x because these do not work. Also, for such procedures as APPEND, SQL, or INSERT, the existing delimiter of the base table—the one being appended to—overrides any specified value for the DELIMITER= option. Otherwise, data corruption would result because the original and appended parts of the resulting table would use different delimiters.

**HAWQ Specifics**
The default is the pipe symbol (`|`).

**Netezza Specifics**
You can use any 7-bit ASCII character as a delimiter. The default is the pipe symbol (`|`). To use a printable ASCII character, enclose it in quotation marks (for example, `BL_DELIMITER="|"`). However, to use an extended character, use the three-digit decimal number representation of the ASCII character for this option. For example, set `BL_DELIMITER=202` to use ASCII character 202 as a delimiter. You must specify decimal number delimiters as three digits even if the first two digits would be zero. For example, specify `BL_DELIMITER=003`, not `BL_DELIMITER=3` or `BL_DELIMITER=03`.

**PostgreSQL Specifics**
The default is a comma (`,`).

**Sybase IQ Specifics**
The default is the pipe symbol (`|`). You can specify the delimiter as a single printable character (such as `|`), or you can use hexadecimal notation to specify any single 8-bit hexadecimal ASCII code. For example, you can specify `BL_DELIMITER="\x09"` to use the tab character as a delimiter.

**Examples**

**Example 1: Specify the Default Delimiter**
Data in this example contains the pipe symbol.

```sas
data work.testdel;
col1='my|data';col2=12;
run;
```
Example 2: Override the Default Delimiter
This example shows how you can override this default when BULKLOAD=YES.

/* Use a comma to delimit data */
proc append base=netlib.mydat(BULKLOAD=YES BL_DELIMITER=',')
data=work.testdel;
run;

Example 3: Override the Default Hadoop Delimiter

data db.joeapp (delim=007); set db.JoeTable2; run;
data db.joeapp (delim="127") ; set db.JoeTable2; run;
data db.joeapp (delimiter=#) ; set db.JoeTable2; run;
data db.joeapp (delimit="#") ; set db.JoeTable2; run;

proc sql;
create table db.joeapp (delim='#') as select * from db.JoeTable2;
quit;

BL_DIRECT_PATH= Data Set Option
Sets the Oracle SQL*Loader DIRECT option.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: YES
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: Oracle
See: BULKLOAD= data set option

Syntax

BL_DIRECT_PATH=YES | NO

Syntax Description

YES
sets the Oracle SQL*Loader option DIRECT to TRUE, letting the SQL*Loader use Direct Path Load to insert rows into a table.

NO
sets the Oracle SQL*Loader option DIRECT to FALSE, letting the SQL*Loader use Conventional Path Load to insert rows into a table.

Details

Conventional Path Load reads in multiple data records and places them in a binary array. When the array is full, it is passed to Oracle for insertion and Oracle uses the SQL interface with the array option.

Direct Path Load creates data blocks that are already in the Oracle database block format. Blocks are then written directly into the database. This method is significantly faster, but there are restrictions. For more information about the SQL*Loader Direct and Conventional Path loads, see your Oracle utilities documentation for SQL*Loader.
BL_DISCARDFILE= Data Set Option

Identifies the file that contains records that were filtered from bulk loading because they did not match the criteria as specified in the CONTROL file.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: creates a file in the current directory or with the default file specifications

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: Oracle

See: BL_BADFILE= data set option [to set the name and location of the file that contains rejected rows], BULKLOAD= data set option

Syntax

BL_DISCARDFILE=path-and-discard-filename

Syntax Description

path-and-discard-filename

- an SQL*Loader discard file containing rows that did not meet the specified criteria.
- On most platforms, the default filename takes the form BL_<table>_<unique-ID>.dsc:

  table
  - specifies the table name

  unique-ID
  - specifies a number that is used to prevent collisions in the event of two or more simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates the number.

Details

SQL*Loader creates the file of discarded rows only if there are discarded rows and if a discard file is requested. If you do not specify this option and a discard file does not exist, a discard file is created in the current directory (or with the default file specifications). If you do not specify this option and a discard file already exists, the Oracle bulk loader reuses the existing file and replaces the contents with discarded rows from the new load.

On most operating systems, the discard file has the same format as the data file, so discarded records can be loaded after corrections are made.

Operating Environment Information

On z/OS operating systems, the discard file is created with default DCB attributes. For information about how to overcome such a case, see the section about SQL*Loader file attributes in the SQL*Loader chapter in the Oracle user's guide for z/OS.
### BL_DISK_PARALLELISM= Data Set Option

Specifies the number of processes or threads to use when writing data to disk.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Requirement</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source</td>
<td>DB2 under UNIX and PC Hosts</td>
</tr>
<tr>
<td>See</td>
<td>BL_CPU_PARALLELISM= data set option, BL_DATA_BUFFER_SIZE= data set option, BULKLOAD= set option</td>
</tr>
</tbody>
</table>

**Syntax**

```plaintext
BL_DISK_PARALLELISM=number-of-processes-or-threads
```

**Syntax Description**

*number of processes or threads*

specifies the number of processes or threads that the load utility uses to write data records to the table-space containers.

**Details**

This option exploits the available containers when it loads data and significantly improves load performance.

The maximum number that is allowed is the greater of 50 or four times the `BL_CPU_PARALLELISM` value, which the load utility actually uses. By default, `BL_DISK_PARALLELISM` is equal to the sum of the table-space containers on all table spaces that contain objects for the table that is being loaded. However, this value cannot exceed the maximum allowed value.

If you do not specify a value, the utility selects an intelligent default that is based on the number of table-space containers and the characteristics of the table.

For more information about using this option, see the DISK_PARALLELISM parameter in the *IBM DB2 Universal Database Data Movement Utilities Guide and Reference*.

---

### BL_ENCODING= Data Set Option

Specifies the character set encoding to use for the external table.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>Requirement</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source</td>
<td>Greenplum, HAWQ</td>
</tr>
<tr>
<td>Note</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See</td>
<td>BULKLOAD= data set option</td>
</tr>
</tbody>
</table>
**Syntax**

`BL_ENCODING=character-set-encoding`

**Syntax Description**

*character-set-encoding*

specifies the character set encoding to use for the external table. Specify a string constant (such as 'SQL_ASCII'), an integer-encoding number, or DEFAULT to use the default client encoding.

---

**BL_ESCAPE= Data Set Option**

Specifies the single character to use for C escape sequences.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** \ (backslash)
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Aster, Greenplum, HAWQ, PostgreSQL
- **Note:** This data set option was new in the first maintenance release for SAS 9.4. Support for Aster was added in the second maintenance release for SAS 9.4.
- **See:** BULKLOAD= data set option

**Syntax**

`BL_ESCAPE='\<any-single-character>' | 'OFF'`

**Details**

Use this option to specify the single character to use for character escape sequences. These can be ‘\n’, ‘\t’, or ‘\001’ (start of heading). It can also be for escape data characters that might otherwise be used as row or column delimiters. Be sure to choose one that is not used anywhere in your actual column data.

Although the default is \ (backslash), you can specify any other character. You can also specify ‘OFF’ to disable the use of escape characters. This is very useful for web log data that contains numerous embedded backslashes that are not intended as escape characters.

*Note:* For Greenplum, you can specify `BL_ESCAPE='\000'` (empty string within single quotation marks) to disable the use of escape characters. This functions in the same way as `BL_ESCAPE='OFF'`.

For octal codes, specify a backslash followed by a three-digit code that represents your escape character, such as ‘\041’ (‘!’). In the third maintenance release for SAS 9.4, octal notation that uses a preceding ‘E’, such as E\024, is not supported. Some common escape characters and their corresponding octal codes are listed in the following table.

<table>
<thead>
<tr>
<th>Escape Character</th>
<th>Octal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(start of heading)</td>
<td>\001</td>
</tr>
</tbody>
</table>

---

**Table 12.1 Octal Codes for Common Escape Characters**
Escape Character | Octal Code
---|---
(device control 4) | \024
! | \041
/ | \057
^ | \136
~ | \176

**BL_EXCEPTION= Data Set Option**

Specifies the exception table into which rows in error are copied.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** none

**Requirement:** To specify this option, you must set BULKLOAD=YES, and you must specify values for BL_REJECT_TYPE= and BL_REJECT_LIMIT= data set options.

**Data source:** DB2 under UNIX and PC Hosts, Greenplum, HAWQ

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** BL_REJECT_LIMIT= data set option, BL_REJECT_TYPE= data set option, BULKLOAD= data set option, Capturing Bulk-Load Statistics into Macro Variables

**Syntax**

```
BL_EXCEPTION=exception-table-name
```

**Syntax Description**

`exception table-name`

specifies the exception table into which rows in error are copied.

**Details**

*DB2 under UNIX and PC Hosts:* Any row that is in violation of a unique index or a primary key index is copied. DATALINK exceptions are also captured in the exception table. If you specify an unqualified table name, the table is qualified with the CURRENT SCHEMA. Information that is written to the exception table is not written to the dump file. In a partitioned database environment, you must define an exception table for those partitions on which the loading table is defined. However, the dump file contains rows that cannot be loaded because they are not valid or contain syntax errors.

For more information about using this option, see the FOR EXCEPTION parameter in IBM DB2 Universal Database Data Movement Utilities Guide and Reference. For more information about the load exception table, see the load exception table topics in IBM DB2 Universal Database Data Movement Utilities Guide and Reference and IBM DB2 Universal Database SQL Reference, Volume 1.
Greenplum, HAWQ: Formatting errors are logged when running in single-row, error-isolation mode. You can then examine this error table to determine whether any error rows were not loaded. The specified error table is used if it already exists. Otherwise, it is generated automatically.

### BL_EXECUTE_CMD= Data Set Option

Specifies the operating system command for segment instances to run.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Restriction:** Only for web tables
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Greenplum, HAWQ
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** BL_EXECUTE_LOCATION= data set option, BL_EXTERNAL_WEB= data set option, BULKLOAD= set option

#### Syntax

```plaintext
BL_EXECUTE_CMD=command | script
```

#### Syntax Description

- **command**
  - specifies the operating system command for segment instances to run.

- **script**
  - specifies a script that contains one or more operating system commands for segment instances to run.

#### Details

Output is web table data at the time of access. Web tables that you define with an EXECUTE clause run the specified script or shell command on the specified hosts. By default, all active segment instances on all segment hosts run the command. For example, if each segment host runs four primary segment instances, the command is executed four times per host. You can also limit the number of segment instances that execute the command.

### BL_EXECUTE_LOCATION= Data Set Option

Specifies which segment instances runs the given command.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Greenplum, HAWQ
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
**BL_EXECUTE_CMD= data set option**

**BL_EXTERNAL_WEB= data set option**

**BL_LOCATION= data set option**

**BULKLOAD= data set option**

### Syntax

**BL_EXECUTE_LOCATION=**

- `ALL`
- `MASTER`
- `HOST [segment-host], number-of-segments`
- `SEGMENT segmentID`

### Syntax Description

**ALL**

specifies that all segment instances run the given command or script.

**MASTER**

specifies that the master segment instance runs the given command or script.

**HOST [segment-hostname], number-of-segments**

indicates that the specified number of segments on the specified host runs the given command or script.

**SEGMENT segmentID**

indicates that the specified segment instance runs the given command or script.

### Details

For more information about valid values for this option, see the documentation for your database engine.

---

**BL_EXTERNAL_WEB= Data Set Option**

Specifies whether the external data set accesses a dynamic data source.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** NO

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** Greenplum, HAWQ

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** Accessing Dynamic Data in Web Tables, BL_EXECUTE_CMD= data set option, BL_EXECUTE_LOCATION= data set option, BULKLOAD= data set option

### Syntax

**BL_EXTERNAL_WEB=**

- `YES`
- `NO`

### Syntax Description

**YES**

specifies that the external data set is not a dynamic data source that resides on the web.
NO
specifies that the external data set is a dynamic data source that resides on the web.

Details
The external data set can access a dynamic data source on the web, or it can run an
operating system command or script. For more information about external web tables,
see the documentation for your database engine.

Example

```sas
libname sasflt 'SAS-library';
libname mydblib greenplm user=myusr1 password=mypwd1 dsn=mysrv1;
proc sql;
create table mydblib.flights98
  (bulkload=yes
   bl_external_web='yes'
   bl_execute_cmd='/var/load_scripts/get_flight_data.sh'
   bl_execute_location='HOST'
   bl_format='TEXT'
   bl_delimiter='|')
  as select * from _NULL_;
quit;
libname sasflt 'SAS-library';
libname mydblib greenplm user=myusr1 password=mypwd1 dsn=mysrv1;
proc sql;
create table mydblib.flights98
  (bulkload=yes
   bl_external_web='yes'
   bl_location_protocol='http'
   bl_datafile='intranet.company.com/expense/sales/file.csv'
   bl_format='CSV')
  as select * from _NULL_;
quit;
```

BL_FILE_BADFILE= Data Set Option
Identifies a file that contains records that were rejected during bulk loading.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Data Access</td>
</tr>
<tr>
<td>Aliases:</td>
<td>BL_ERRORFILE</td>
</tr>
<tr>
<td></td>
<td>BL_BADFILE</td>
</tr>
<tr>
<td></td>
<td>BL_FILE_BADFILE</td>
</tr>
<tr>
<td></td>
<td>BL_FILE_LOG</td>
</tr>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Restriction:</td>
<td>For use with a distributed system, this filename must point to a shared disk.</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>SAP HANA</td>
</tr>
</tbody>
</table>
Note: Use this option to specify the fully qualified filename for the temporary error file.

See: BL_CONTROLFIELDDELIMITER= data set option,
BL_CONTROLQUOTATIONMARK= data set option,
BL_CONTROLRECORDDELIMITER= data set option,
BL_FILECONTROLFILE= data set option,
BL_FILEBADFILE= data set option,
BL_FILEDATAFILE= data set option,
BL_FILEDEFAULT_DIR= data set option,
BL_FILEDELETE_DATAFILE= data set option,
BULKLOAD= data set option

Syntax

BL_FILEBADFILE=default-dir/BL_table_name_number.err

BL_FILECONTROLFILE= Data Set Option

Identifies the file that contains control statements.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Access
Aliases: BL_CONTROL
BL_CONTROLFILE
BL_FILECONTROLFILE

Default: default-dir/BL_table_name_number.ctl

Restriction: For use with a distributed system, this filename must point to a shared disk.
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: SAP HANA

Note: Use this option to specify the fully qualified filename for the temporary control file.

See: BL_CONTROLFIELDDELIMITER= data set option,
BL_CONTROLQUOTATIONMARK= data set option,
BL_CONTROLRECORDDELIMITER= data set option,
BL_FILECONTROLFILE= data set option,
BL_FILEBADFILE= data set option,
BL_FILEDATAFILE= data set option,
BL_FILEDEFAULT_DIR= data set option,
BL_FILEDELETE_DATAFILE= data set option,
BULKLOAD= data set option

Syntax

BL_FILECONTROLFILE=default-dir/BL_number.ctl

Details

The SAP HANA interface creates the control file by using information from the input data and SAS/ACCESS options. The file contains Data Definition Language (DDL) definitions that specify the location of the data and how the data corresponds to the database table. It is used to specify exactly how the loader should interpret the data that you are loading from the DATA file (.dat file). By default it creates a control file in the current directory or with the default file specifications.

If you do not specify this option and a control file does not already exist, a file is created in the current directory or with the default file specifications. If you do not specify this option and a control file already exists, the Oracle interface reuses the file and replaces the contents with the new control statements.
### BL_FILE_DATAFILE= Data Set Option

Identifies the file that contains DBMS data for bulk loading.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Access
- **Aliases:** BL_DATAFILE
  BL_FILE_DATAFILE
- **Default:** none
- **Restriction:** For use with a distributed system, this filename must point to a shared disk.
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** SAP HANA

**Note:** Use this option to specify the fully qualified filename for the temporary data file.

**See:**
BL_CONTROL_FIELD_DELIMITER= data set option,
BL_CONTROL_QUOTATION_MARK= data set option,
BL_CONTROL_RECORD_DELIMITER= data set option, BL_FILE_BADFILE= data set option, BL_FILE_CONTROLFILE= data set option, BL_DELETE_DATAFILE= data set option,
BL_FILE_DEFAULT_DIR= data set option,
BL_FILE_DELETE_DATAFILE= data set option, BULKLOAD= data set option

---

### Syntax

\[\text{BL\_FILE\_DATAFILE}=\text{default-dir}/\text{BL\_table\_name\_number.dat}\]

### Details

By default, the SAP HANA interface creates a data file with a DAT file extension in the current directory or with the default file specifications. In addition, by default, the data file is deleted after the load is completed. To override this behavior, specify BL_DELETE_DATAFILE=NO.

---

### BL_FILE_DEFAULT_DIR= Data Set Option

Specifies the default directory for the temporary files to use in bulk loading.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Access
- **Aliases:** BL_DEFAULT_DIR
  BL_FILE_DEFAULT_DIR
- **Default:** '/tmp'
- **Requirements:** To specify this option, you must first set BULKLOAD=YES.
  When you use a distributed system, the path must point to a shared disk.
- **Data source:** SAP HANA
BL_FILE_DEFAULT_DIR= Data Set Option

Specifies whether to delete only the data file or all files that the SAS/ACCESS engine creates for the DBMS bulk-load facility.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Access
Aliases: BL_DELETE_DATAFILE
BL_FILE_DELETE_DATAFILE
Default: YES

Syntax

```
BL_FILE_DEFAULT_DIR='filepath'
```

BL_FILE_DELETE_DATAFILE= Data Set Option

Specifies whether to delete only the data file or all files that the SAS/ACCESS engine creates for the DBMS bulk-load facility.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Access
Aliases: BL_DELETE_DATAFILE
BL_FILE_DELETE_DATAFILE
Default: YES

Syntax Description

```
BL_FILE_DELETE_DATAFILE=YES | NO
```

YES deletes the data file that the SAS/ACCESS engine creates for the DBMS bulk-load facility.

NO does not delete the file.

BL_FORCE_NOT_NULL= Data Set Option

Specifies how to process CSV column values.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: Aster, Greenplum, HAWQ

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: BL_DELIMITER= data set option, BL_FORMAT= data set option, BL_NULL= data set option, BL_QUOTE= data set option, BULKLOAD= data set option

---

**Syntax**

BL_FORCE_NOT_NULL= **YES | NO**

**Syntax Description**

**YES**

specifies that each specified column is processed as if it is enclosed in quotation marks and is therefore not a null value.

**NO**

specifies that each specified column is processed as if it is a null value.

**Details**

You can use this option only when BL_FORMAT=CSV. For the default null string, where no value exists between two delimiters, missing values are evaluated as zero-length strings.

---

**BL_FORMAT= Data Set Option**

Specifies the format of the external or web table data.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** TEXT

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** Greenplum, HAWQ, PostgreSQL

**Notes:** Support for this data set option was added in the first maintenance release for SAS 9.4.

Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** BL_DELIMITER= data set option, BL_FORCE_NOT_NULL= data set option, BL_NULL= data set option, BL_QUOTE= data set option, BULKLOAD= data set option

---

**Syntax**

BL_FORMAT= **TEXT | CSV**

**Syntax Description**

**TEXT**

specifies plain text format.
CSV
   specifies a comma-separated value format.

**BL_HEADER= Data Set Option**
Indicates whether to skip or load the first record in the input data file.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Restriction:** You can use this option only when loading a table using an external web source.
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Greenplum, HAWQ
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_HEADER=YES | NO
```

**Syntax Description**

**YES**
   indicates that the first record is skipped (not loaded).

**NO**
   indicates that the first record is loaded.

**Details**

When the first record of the input data file contains the name of the columns to load, you can indicate that it should be skipped during the load process.

**BL_HOST= Data Set Option**

Specifies the host name or IP address of the server where the external data file is stored.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** DBMS-specific; SERVER= value, if SERVER= is set; otherwise, none [Impala]
- **Requirements:** To specify this option, you must first set BULKLOAD=YES.
   You must enclose the name in quotation marks.
- **Data source:** Aster, Greenplum, HAWQ, Impala
- **Note:** Support for Impala was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** BL_DBNAME= data set option, BL_PATH= data set option, BULKLOAD= data set option
Syntax
BL_HOST='hostname' [Aster, Impala]
BL_HOST='localhost' [Greenplum, HAWQ]

Syntax Description

localhost
specifies the IP address of the server where the external data file is stored.

Details

Use this option to pass the IP address to the DBMS bulk-load facility.

Greenplum: The default is 127.0.0.1. You can use the GPLOAD_HOST environment variable to override the default.

BL_IMPORT_BATCH_SIZE= Data Set Option

Specifies the number of records to insert in each commit.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Access
Aliases: BL_BATCH_SIZE
BL_IMPORT_BATCH_SIZE
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: SAP HANA
See: BL_IMPORT_BATCH_SIZE=, BL_IMPORT_OPTIONS=, BL_IMPORT_TABLE_LOCK=, BL_IMPORT_TYPE_CHECK=, BULKLOAD= data set option

Syntax

BL_IMPORT_BATCH_SIZE=batch-size

BL_IMPORT_OPTIONS= Data Set Option

Specifies additional options to add to the SAP HANA IMPORT statement.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Access
Aliases: BCP_OPTIONS
BLIMPORT_OPTIONS
BL_MODIFIED_BY
BL_OPTIONS
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: SAP HANA

Tip: Options are appended at the end of the generated IMPORT statement.

See: BL_IMPORT_BATCH_SIZE=, BL_IMPORT_TABLE_LOCK=, BL_IMPORT_TYPE_CHECK=, BULKLOAD= data set option

---

**Syntax**

BL_IMPORT_OPTIONS=\*options\*

---

**BL_IMPORT_TABLE_LOCK= Data Set Option**

Specifies whether to lock the table for data import into column store tables.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Category:** Data Access

**Aliases:** BL_IMPORT_TABLE_LOCK

BL_TABLE_LOCK

**Default:** NO

**Restriction:** If you do not set this option, no locking occurs.

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** SAP HANA

See: BL_IMPORT_BATCH_SIZE=, BL_IMPORT_OPTIONS=, BL_IMPORT_TYPE_CHECK=, BULKLOAD= data set option

---

**Syntax**

BL_IMPORT_TABLE_LOCK=YES \| NO

**Syntax Description**

**YES**

specifies to add the TABLE LOCK option to the SAP HANA IMPORT statement.

**NO**

specifies that no locking occurs.

---

**BL_IMPORT_TYPE_CHECK= Data Set Option**

Specifies whether to insert the record without checking the data type of each field.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Category:** Data Access

**Aliases:** BL_IMPORT_TYPE_CHECK

BL_TYPE_CHECK
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: SAP HANA
See: BL_IMPORT_BATCH_SIZE=, BL_IMPORT_OPTIONS=, BL_IMPORT_TABLE_LOCK=, BULKLOAD= data set option

Syntax

BL_IMPORT_TYPE_CHECK= YES | NO

Syntax Description

YES
specifies to add the NO TYPE CHECK option to the SAP HANA IMPORT statement.

NO
specifies that no checking of data types occurs.

BL_INDEX_OPTIONS= Data Set Option

Lets you specify SQL*Loader Index options with bulk loading.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Alias: SQLLDR_INDEX_OPTION=

Default: none

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: Oracle

See: BULKLOAD= data set option

Syntax

BL_INDEX_OPTIONS= any valid SQL*Loader Index option segment-name

Syntax Description

any valid SQL*Loader Index option
The value that you specify for this option must be a valid SQL*Loader index option, such as one of those below. Otherwise, an error occurs.

SINGLEROW
Use this option when loading either a direct path with APPEND on systems with limited memory or a small number of records into a large table. It inserts each index entry directly into the index, one record at a time. By default, DQL*Loader does not use this option to append records to a table.

SORTED INDEXES
This clause applies when you are loading a direct path. It tells the SQL*Loader that the incoming data has already been sorted on the specified indexes, allowing SQL*Loader to optimize performance. It lets the SQL*Loader optimize index
creation by eliminating the sort phase for this data when using the direct-path load method.

Details

You can now pass in SQL*Loader index options when bulk loading. For details about these options, see the Oracle utilities documentation.

Example: Specify SQL*Loader Index Options

This example shows how you can use this option.

```sql
proc sql;
connect to oracle ( user=myusr1 pw=mypwd1 path=mypath);
execute ( drop table blidxopts) by oracle;
execute ( create table blidxopts ( empno number, empname varchar2(20))) by oracle;
execute ( drop index blidxopts_idx) by oracle;
execute ( create index blidxopts_idx on blidxopts ( empno ) ) by oracle;
quit;
libname x oracle user=myusr1 pw=mypwd1 path=mypath;
data new;
empno=1; empname='one';
output;
empno=2; empname='two';
output;
run;
proc append base= x.blidxopts( bulkload=yes bl_index_options='sorted indexes ( blidxopts_idx)' ) data= new;
run;
```

BL_INDEXING_MODE= Data Set Option

Indicates which scheme the DB2 load utility should use for index maintenance.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** AUTOSELECT
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under UNIX and PC Hosts
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_INDEXING_MODE= AUTOSELECT | REBUILD | INCREMENTAL | DEFERRED
```

**Syntax Description**

**AUTOSELECT**

The load utility automatically decides between REBUILD or INCREMENTAL mode.
REBUILD
   All indexes are rebuilt.

INCREMENTAL
   Indexes are extended with new data.

DEFERRED
   The load utility does not attempt index creation if this mode is specified. Indexes are marked as needing a refresh.

Details
For more information about using the values for this option, see the IBM DB2 Universal Database Data Movement Utilities Guide and Reference.

---

**BL_KEEPIDENTITY= Data Set Option**

Determines whether the identity column that is created during bulk loading is populated with values that Microsoft SQL Server generates or with values that the user provides.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** LIBNAME option setting

**Restriction:** This option is valid only when you use the Microsoft SQL Server provider.

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** OLE DB

**See:** BL_KEEPIDENTITY= LIBNAME option, BULKLOAD= data set option

**Syntax**

```
BL_KEEPIDENTITY=YES | NO
```

**Syntax Description**

**YES**
- specifies that the user must provide values for the identity column.

**NO**
- specifies that the Microsoft SQL Server generates values for an identity column in the table.

---

**BL_KEEPNULLS= Data Set Option**

Indicates how NULL values in Microsoft SQL Server columns that accept NULL are handled during bulk loading.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** LIBNAME option setting

**Restriction:** This option affects values in only Microsoft SQL Server columns that accept NULL and that have a DEFAULT constraint.

**Requirement:** To specify this option, you must first set BULKLOAD=YES.
Data source: OLE DB

See: BL_KEEPNULLS= LIBNAME option, BULKLOAD= data set option

**Syntax**

**BL_KEEPNULLS**=YES | NO

**Syntax Description**

**YES**

preserves null values that the OLE DB interface inserts.

**NO**

replaces null values that the OLE DB interface inserts with a default value, as specified in the DEFAULT constraint.

---

**BL_LOAD_METHOD= Data Set Option**

Specifies the method by which data is loaded into an Oracle or PostgreSQL table during bulk loading.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Category:** Bulk Loading

**Default:** INSERT when loading an empty table; APPEND when loading a table that contains data

**Restriction:** REPLACE and TRUNCATE values apply only when you are loading data into a table that already contains data. In this case, you can use REPLACE and TRUNCATE to override the default value of APPEND. See your Oracle or PostgreSQL utilities documentation for information about using the TRUNCATE and REPLACE load methods.

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** Oracle, PostgreSQL

**See:** BULKLOAD= data set option

---

**Syntax**

**BL_LOAD_METHOD**=INSERT | APPEND | REPLACE | TRUNCATE

**BL_LOAD_METHOD**=APPEND | REPLACE | TRUNCATE

**Syntax Description**

**INSERT**

requires the DBMS table to be empty before loading. (only Oracle)

**APPEND**

appends rows to an existing DBMS table.

**REPLACE**

deletes all rows in the existing DBMS table and loads new rows from the data file.
TRUNCATE uses the SQL truncate command to achieve the best possible performance. You must first disable the referential integrity constraints of the DBMS table.

**BL_LOAD_REPLACE= Data Set Option**

Specifies whether DB2 appends or replaces rows during bulk loading.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** DB2 under UNIX and PC Hosts
- **See:** BULKLOAD= data set option

**Syntax**

BL_LOAD_REPLACE=YES | NO

**Syntax Description**

- **NO**
  - the CLI LOAD interface appends new rows of data to the DB2 table.
- **YES**
  - the CLI LOAD interface replaces the existing data in the table.

**BL_LOCATION= Data Set Option**

Specifies the location of a file on a web server for segment hosts to access.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Greenplum, HAWQ
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** BL_EXECUTE_LOCATION= data set option, BL_HOST= data set option, BULKLOAD= data set option

**Syntax**

BL_LOCATION=http://file-location

**BL_LOG= Data Set Option**

Identifies a log file that contains information for bulk loading, such as statistics and errors.
Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: DBMS-specific

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: DB2 under UNIX and PC Hosts, Oracle, Teradata

See: BL_LOG= LIBNAME option, BULKLOAD= data set option, DB2 under UNIX and PC Hosts Bulk Loading, Maximizing Teradata Load Performance (Teradata bulk loading)

Syntax

BL_LOG=path-and-log-filename

Syntax Description

path-and-log-filename

specifies a file where information about the loading process is written.

Details

See the reference section for your SAS/ACCESS interface for additional details about specifying this option.

When the DBMS bulk-load facility is invoked, it creates a log file. The contents of the log file are DBMS-specific. The BL_ prefix distinguishes this log file from the one created by the SAS log. If BL_LOG= is specified with the same path and filename as an existing log, the new log replaces the existing log.

DB2 under UNIX and PC Hosts: If BL_LOG= is not specified, the log file is deleted automatically after a successful operation. For more information, see the bulk-load topic in the DB2 under UNIX and PC Hosts bulk-load section.

Teradata: For more information, see the bulk-load topic for Teradata.

Oracle: When the SQL*Loader is invoked, it creates a log file. This file contains a detailed summary of the load, including a description of any errors. If SQL*Loader cannot create a log file, execution of the bulk loading terminates. If a log file does not already exist, it is created in the current directory or with the default file specifications. If a log file does already exist, the Oracle bulk loader reuses the file, replacing the contents with information from the new load. On most platforms, the default filename takes the form BL_<table>_<unique-ID>.log:

table

specifies the table name

unique-ID

specifies a number that is used to prevent collisions in the event of two or more simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates the number.
**BL_LOGFILE= path-and-log-filename**

**Syntax Description**

*path-and-log-filename* specifies a file where information about the loading process is written.

---

**BL_MAPFILE= Data Set Option**

Indicates whether to use a map file to load or export tables.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Requirement:** To specify this option, you must first set BULKLOAD=YES or BULKUNLOAD=YES.
- **Data source:** Aster
- **Note:** Support for this data set option was added for SAS 9.4.
- **See:** BULKLOAD= LIBNAME option, BULKLOAD= data set option, BULKUNLOAD= LIBNAME option, BULKUNLOAD= data set option

**Syntax**

BL_MAPFILE=YES | NO

**Syntax Description**

- **YES** specifies that a map file is used.
- **NO** specifies that a map file is not used.

**Details**

You can use this option to pass connection and table information in a text (map) file to the loader or export application during bulk loading or unloading. The SAS/ACCESS engine automatically creates and deletes the map file for each bulk loading or unloading.

**Example: Passing Connection and Table Information**

SAS uses the sasflt.flt98 map file in this example to provide the needed connection and table information for bulk loading.

```plaintext
LIBNAME sasflt 'SAS-data-library';
```
LIBNAME net_air ASTER user=myusr1 pwd=mypwd1
server=air2 database=flights dimension=yes;

data net_air.flights98
  (bulkload=YES bl_mapfile=yes);
set sasflt.flt98;
run;

**BL_METHOD= Data Set Option**

Specifies the bulk-loading method to use for DB2.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Requirement:</td>
<td>Before you can use the CLI LOAD interface, you must first set BULKLOAD=YES and then specify this option.</td>
</tr>
</tbody>
</table>
| Data source: | DB2 under UNIX and PC Hosts | See: BULKLOAD= data set option

**Syntax**

```
BL_METHOD=CLLOAD
```

**Syntax Description**

*CLLOAD* enables the CLI LOAD interface to the LOAD utility.

---

**BL_NULL= Data Set Option**

Specifies the string that represents a null value.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>&quot;\N&quot; [TEXT mode], unquoted empty value [CSV mode]</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>Greenplum, HAWQ, PostgreSQL</td>
</tr>
<tr>
<td>See:</td>
<td>BL_DELIMITER= data set option, BL_FORCE_NOT_NULL= data set option, BL_FORMAT= data set option, BL_QUOTE= data set option, BULKLOAD= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

```
BL_NULL="\N" | empty-value
```
Details

You might prefer an empty string even in TEXT mode for cases where you do not want to distinguish nulls from empty strings. When you use this option with external and web tables, any data item that matches this string is considered a null value.

BL_OPTIONS= Data Set Option

Passes options to the DBMS bulk-load facility, which affects how it loads and processes data.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: DBMS-specific
Requirements: To specify this option, you must first set BULKLOAD=YES.
You must separate multiple options with commas and enclose the entire string of options in single quotation marks.
Data source: Aster, DB2 under UNIX and PC Hosts, Netezza, OLE DB, Oracle, Sybase IQ
See: BL_OPTIONS= LIBNAME option, BULKLOAD= LIBNAME option, BULKLOAD= data set option

Syntax

BL_OPTIONS='option-1<,option-2<,...,option-N>>' [DB2 under UNIX and PC Hosts, OLE DB, Oracle]
BL_OPTIONS='option-1 value-1<,option-2 value-2<... option-N value-N>>' [Aster, Netezza, Sybase IQ]

Syntax Description

option-N specifies an option from the available options that are specific to each SAS/ACCESS interface.

Details

You can use BL_OPTIONS= to pass options to the DBMS bulk-load facility when it is called, which affects how data is loaded and processed.

Aster: By default, no options are specified.

DB2 under UNIX and PC Hosts: This option passes DB2 file-type modifiers to DB2 LOAD or IMPORT commands to affect how data is loaded and processed. Not all DB2 file type modifiers are appropriate for all situations. You can specify one or more DB2 file type modifiers with IXF files. For a list of file type modifiers, see the description of the LOAD and IMPORT utilities in the IBM DB2 Universal Database Data Movement Utilities Guide and Reference.

Netezza: Any text that you enter for this option is appended to the USING clause of the CREATE EXTERNAL TABLE statement—namely, any external_table_options in the Netezza Database User's Guide.

OLE DB: By default, no options are specified. This option is valid only when you are using the Microsoft SQL Server provider. This option takes the same values as the -h HINT option of the Microsoft BCP utility. For example, the ORDER= option sets the sort order of data in the data file; you can use it to improve performance if the file is sorted according to the clustered index on the table. See the Microsoft SQL Server documentation for a complete list of supported bulk copy options.
**Oracle:** This option lets you specify the SQL*Loader options ERRORS= and LOAD=. The ERRORS= option specifies the number of insert errors that terminates the load. The default value of ERRORS=1000000 overrides the default value for the Oracle SQL*Loader ERRORS= option, which is 50. LOAD= specifies the maximum number of logical records to load. If the LOAD= option is not specified, all rows are loaded. See your Oracle utilities documentation for a complete list of SQL*Loader options that you can specify in BL_OPTIONS=.

**Sybase IQ:** By default, no options are specified. Any text that you enter for this option is appended to the LOAD TABLE command that the SAS/ACCESS interface uses for the bulk-loading process.

**Examples**

**Example 1: Specify the Number of Permitted Errors**

In this Oracle example BL_OPTIONS= specifies the number of errors that are permitted during a load of 2,000 rows of data, where all listed options are enclosed in quotation marks.

```plaintext
bl_options='ERRORS=999,LOAD=2000'
```

**Example 2: Specify External Table Options**

This Netezza example shows you how to use BL_OPTIONS= to specify two different external table options, ctrlchars and logdir:

```plaintext
data netlib.mdata(bulkload=yes bl_options="ctrlchars true logdir 'c:\temp'";
set saslib.transdata;
run;
```

---

**BL_PARFILE= Data Set Option**

Creates a file that contains the SQL*Loader command-line options.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Oracle
- **Tip:** The parse file is deleted at the end of SQL*Loader processing.
- **See:** BULKLOAD= data set option

**Syntax**

```plaintext
BL_PARFILE=<parse-file>
```

**Syntax Description**

- **parse-file**

  the name that you give the file that contains the SQL*Loader command line options. It can also specify the path. If you do not specify a path, the file is created in the current directory.
Details

This option prompts the SQL*Loader to use the PARFILE= option. This SQL*Loader option lets you specify SQL*Loader command-line options in a file instead of as command-line options. Here is an example of how you can call the SQL*Loader by specifying user ID and control options.

```sql
sqlldr userid=myusr1/mypwd1 control=example.ctl
```

You can also call it by using the PARFILE= option.

```sql
sqlldr parfile=example.par
```

Example.par now contains the USERID= and CONTROL= options. Security is a major advantage of using the BL_PARFILE= option because the user ID and password are stored in a separate file.

Permissions on the file default to operating system defaults. Create the file in a protected directory to prevent unauthorized users from accessing its contents.

To display the contents of the parse file in the SAS log, use the `SASTRACE=",,,d"` option. The password is blocked out and replaced with `xxxx`, however.

Example: Call the SQL*Loader without and with BL_PARFILE=

This example demonstrates how SQL*Loader invocation differs when you specify the BL_PARFILE= option.

```sas
libname x oracle user=myusr1 pw=mypwd1;

/* In this DATA step, call the SQL*Loader without BL_PARFILE=.  
   sqlldr userid=myusr1/mypwd1@oraclev9  
   control=bl_bltst_0.ctl log=bl_bltst_0.log  
   bad=bl_bltst_0.bad discard=bl_bltst_0.dsc  
*/
data x.bltst ( bulkload=yes);  
c1=1;  
run;

/* In this DATA step, call the SQL*Loader with BL_PARFILE=.  
   sqlldr parfile=test.par
   In this case all options are written to the test.par file.  
*/
data x.bltst2 ( bulkload=yes bl_parfile='test.par');  
c1=1;  
run;
```

BL_PATH= Data Set Option

Specifies the path to use for bulk loading.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** none

**Requirements:** To specify this option, you must first set BULKLOAD=YES.
You must enclose the entire path in quotation marks.

**Data source:** Aster

**See:** BL_DBNAME= data set option, BL_HOST= data set option, BULKLOAD= data set option

---

**Syntax**

BL_PATH='path'

**Syntax Description**

*path*

specifies the path to use for bulk loading.

**Details**

Use this option to pass the path to the DBMS bulk-load facility.

---

**BL_PORT= Data Set Option**

Specifies the port number to use.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Alias:** BULKLOAD_PORT= [Hadoop, Impala], BLPORT= [Impala]

**Default:** 8020 [Hadoop], 8080 [Greenplum, HAWQ], 50070 [Impala]

**Restriction:** Hadoop: This option is required if Hadoop HDFS service is running on a port other than 8020.

**Requirement:** To specify this option, you must first set BULKEXTRACT= YES or BULKLOAD=YES.

**Data source:** Greenplum, Hadoop, HAWQ, Impala

**Note:** Support for Impala was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** BULKLOAD= data set option

---

**Syntax**

BL_PORT=port

**Syntax Description**

*port*

specifies the port number to use.

**Details**

Use this option to specify the port number that bulk loading uses to communicate with the server where the input data file resides.
BL_PORT_MAX= Data Set Option

Sets the highest available port number for concurrent uploads.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirements: To specify this option, you must first set BULKLOAD=YES.
To reserve a port range, you must specify values for this option and also the BL_PORT_MIN= option.
Data source: DB2 under UNIX and PC Hosts
See: BL_PORT_MIN= data set option, BULKLOAD= data set option

Syntax

BL_PORT_MAX=integer

Syntax Description

integer specifies a positive integer that represents the highest available port number for concurrent uploads.

BL_PORT_MIN= Data Set Option

Sets the lowest available port number for concurrent uploads.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirements: To specify this option, you must first set BULKLOAD=YES.
To reserve a port range, you must specify values for both this option and the BL_PORT_MAX= option.
Data source: DB2 under UNIX and PC Hosts
See: BL_PORT_MAX= data set option, BULKLOAD= data set option

Syntax

BL_PORT_MIN=integer

Syntax Description

integer specifies a positive integer that represents the lowest available port number for concurrent uploads.
**BL_PRESERVE_BLANKS= Data Set Option**

Determines how the SQL*Loader handles requests to insert blank spaces into CHAR/VARCHAR2 columns with the NOT NULL constraint.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Restriction:** This option is not supported on z/OS.
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Oracle
- **See:** BULKLOAD= data set option

**Syntax**

```
BL_PRESERVE_BLANKS=YES | NO
```

**Syntax Description**

**YES**

specifies that blank values are inserted as blank spaces.

**CAUTION:**

When this option is set to YES, any trailing blank spaces are also inserted. For this reason, use this option with caution. It is recommended that you set this option to YES only for CHAR columns. Do not set this option to YES for VARCHAR2 columns because trailing blank spaces are significant in VARCHAR2 columns.

**NO**

specifies that blank values are inserted as null values.

---

**BL_PROTOCOL= Data Set Option**

Specifies the protocol to use.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** gpfdist
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Greenplum, HAWQ
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** BL_DATAFILE= data set option, BL_HOST= data set option, BULKLOAD= data set option, Protocols for Accessing External Tables, Using the file:// Protocol

**Syntax**

```
BL_PROTOCOL= gpfdist | file | http
```
Syntax Description

gpfdist
   specifies the Greenplum file distribution program.

file
   specifies external tables on a segment host.

http
   specifies web address of a file on a segment host. This value is valid only for external web tables.

---

**BL_PSQL_PATH= Data Set Option**

Specifies the location of the PSQL executable file.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** psql

**Requirement:** To specify this option, you must first set BULKLOAD=YES.

**Data source:** PostgreSQL

**See:** BULKLOAD= data set option

---

**Syntax**

BL_PSQL_PATH=pathname

**Syntax Description**

pathname
   specifies the full pathname to the PostgreSQL PSQL executable file to let the SAS/ACCESS interface call the PostgreSQL PSQL utility.

---

**BL_QUOTE= Data Set Option**

Specifies the quotation character for CSV mode.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** " (double quote)

**Requirements:** CVS=YES
   MULTILOAD=YES

**Data source:** Aster, Greenplum, HAWQ, PostgreSQL

**Notes:** Support for this data set option was added in the first maintenance release for SAS 9.4.
   Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** BL_DELIMITER= data set option, BL_ESCAPE= data set option, BL_FORCE_NOT_NULL= data set option, BL_FORMAT= data set option, BL_NULL= data set option, BULKLOAD= data set option
**Syntax**

\texttt{BL\_QUOTE=} \textit{single 1-byte character}

**Syntax Description**

\textit{single 1-byte character}

specifies the quoting character to be used when a data value is quoted. This must be a single one-byte character.

**Details**

This option is allowed only when \texttt{BL\_FORMAT=} CSV. The default is a double quotation mark (""). If your data contains the double quotation mark, then set \texttt{BL\_QUOTE=} to a character that is not included in the data.

The characters that are assigned to \texttt{BL\_QUOTE=} and to \texttt{BL\_DELIMITER=} must be different.

---

**\texttt{BL\_RECOVERABLE=} Data Set Option**

Determines whether the LOAD process is recoverable.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS\textslash{}ACCESS software) |
| Default:  | DBMS-specific |
| Requirement: | To specify this option, you must first set BULKLOAD=YES. |
| Data source: | DB2 under UNIX and PC Hosts, Oracle |
| See: | BULKLOAD= data set option |

**Syntax**

\texttt{BL\_RECOVERABLE=} \texttt{YES} \texttt{\mid} \texttt{NO}

**Syntax Description**

\texttt{YES}

specifies that the LOAD process is recoverable. For DB2, YES also specifies that \texttt{BL\_COPY\_LOCATION=} should specify the copy location for the data.

\texttt{NO}

specifies that the LOAD process is not recoverable.

**Details**

\textit{DB2 under UNIX and PC Hosts:} The default is NO.

\textit{Oracle:} The default is YES. Set this option to NO to improve direct load performance. Specifying NO adds the UNRECOVERABLE keyword before the LOAD keyword in the control file.

\textbf{CAUTION:}

Be aware that an unrecoverable load does not log loaded data into the redo log file. Therefore, media recovery is disabled for the loaded table. For more information about the implications of using the UNRECOVERABLE parameter in Oracle, see your Oracle utilities documentation.
Example: Specify a Load as Unrecoverable

This Oracle example shows how to use BL_RECOVERABLE= to specify that the load is unrecoverable.

data x.recover_no (bulkload=yes bl_recoverable=no); c1=1; run;

BL_REJECT_LIMIT= Data Set Option

Specifies the reject limit count.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES and then set BL_REJECT_TYPE=.
Interaction: If the BL_EXCEPTION= data set option is not defined, then rejected records are dropped and are not saved to an exception table.
Data source: Greenplum, HAWQ
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: BL_EXCEPTION= data set option, BL_REJECT_TYPE= data set option, BULKLOAD= data set option

Syntax

BL_REJECT_LIMIT=number

Syntax Description

number 

specifies the reject limit count either as a percentage (1 to 100) of total rows or as a number of rows.

Details

When BL_REJECT_TYPE=PERCENT, the percentage of rows per segment is calculated based on the Greenplum database configuration parameter (gp_reject_percent_threshold). The default value for this parameter Greenplum parameter is 300.

Input rows with format errors are discarded if the reject limit count is not reached on any Greenplum segment instance during the load operation.

Constraint errors result when violations occur to such constraints as NOT NULL, CHECK, or UNIQUE. A single constraint error causes the entire external table operation to fail. If the reject limit is not reached, rows without errors are processed and rows with errors are discarded.

BL_REJECT_TYPE= Data Set Option

Indicates whether the reject limit count is a number of rows or a percentage of total rows.
Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: ROWS
Requirement: To specify this option, you must first set BULKLOAD=YES.
Interaction: If the BL_EXCEPTION= data set option is not defined, then rejected records are dropped and are not saved to an exception table.
Data source: Greenplum, HAWQ
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: BL_REJECT_LIMIT= data set option, BULKLOAD= data set option

Syntax

BL_REJECT_TYPE=ROWS | PERCENT

Syntax Description

ROWS
specifies the reject limit count as a number of rows.

PERCENT
specifies the reject limit count as a percentage of total rows. Valid values range from 1 to 100.

BL_REMOTE_FILE= Data Set Option

Specifies the base filename and location of DB2 LOAD temporary files.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: DB2 under UNIX and PC Hosts
See: BL_SERVER_DATAFILE= data set option (specifies the path from the server), BULKLOAD= data set option, DB2 under z/OS Bulk Loading

Syntax

BL_REMOTE_FILE=pathname-and-base-filename

Syntax Description

pathname-and-base-filename
the full pathname and base filename to which DB2 appends extensions (such as .log, .msg, and .dat files) to create temporary files during load operations. By default, BL_\<table>\_<unique-ID> is the form of the base filename.

\table
specifies the table name.
unique-ID
specifies a number that prevents collisions in the event of two or more simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates this number.

Details
When you specify this option, the DB2 LOAD command is used instead of the IMPORT command. For more information about these commands, see the bulk-load topic in the DB2 under z/OS section.

For pathname, specify a location on a DB2 server that is accessed exclusively by a single DB2 server instance, and for which the instance owner has Read and Write permissions. Make sure that each LOAD command is associated with a unique pathname-and-base-filename value.

BL_RETURN_WARNINGS_AS_ERRORS= Data Set Option
Specifies whether SQL*Loader (bulk-load) warnings should be displayed in SAS through the SYSERR macro as warnings or errors.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: Oracle
See: BULKLOAD= data set option

Syntax
BL_RETURN_WARNINGS_AS_ERRORS=YES | NO

Syntax Description
YES
specifies to return all SQL*Loader warnings as errors, which SYSERR reflects.
NO
specifies to return all SQL*Loader warnings as warnings.

BL_SERVER_DATAFILE= Data Set Option
Specifies the name and location of the data file that the DBMS server instance sees.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Alias: BL_DATAFILE
Default: creates a data file in the current directory or with the default file specifications (same as for BL_DATAFILE=)
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: DB2 under UNIX and PC Hosts, Sybase IQ
BL_CLIENT_DATAFILE= data set option, BL_DATAFILE= data set option, BL_REMOTE_FILE= data set option, BULKLOAD= data set option
“Bulk Loading for DB2 under UNIX and PC Hosts”, Bulk Loading for Sybase IQ

Syntax

BL_SERVER_DATAFILE=path-and-data-filename

Syntax Description

pathname-and-data-filename

specifies the fully qualified pathname and filename of the data file to load, as seen by the DBMS server instance. By default, the base filename takes the form BL_<table>_<unique-ID>:

table

specifies the table name.

unique-ID

specifies a number that is used to prevent collisions in the event of two or more simultaneous bulk loadings of a particular table. The SAS/ACCESS engine generates the number.

Details

*DB2 under UNIX and PC Hosts*: You must also specify a value for BL_REMOTE_FILE=. If the path to the data file from the DB2 server instance is different from the path to the data file from the client, you must use BL_SERVER_DATAFILE= to specify the path from the DB2 server. By enabling the DB2 server instance to directly access the data file that BL_DATAFILE= specifies, this option facilitates use of the DB2 LOAD command. For more information about the LOAD command, see the bulk-load topic in the DB2 under z/OS section. To specify the path from the client, see the BL_DATAFILE= data set option.

Sybase IQ: To specify the path from the client, see the BL_CLIENT_DATAFILE= data set option, which is the client view of the data file.

---

BL_SFTP_HOST= Data Set Option

Specifies the network name of the remote host with the OpenSSH secure shell daemon (SSHD) server running.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Data Access</td>
</tr>
<tr>
<td>Aliases:</td>
<td>BL_HOST</td>
</tr>
<tr>
<td></td>
<td>BL_SERVER</td>
</tr>
<tr>
<td></td>
<td>BL_SFTP_HOST</td>
</tr>
<tr>
<td></td>
<td>BL_SFTP_SERVER</td>
</tr>
<tr>
<td>Default:</td>
<td>SAP-HANA-server-name</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>SAP HANA</td>
</tr>
</tbody>
</table>
See: BL_SFTP_OPTIONS= data set option, BL_SFTP_USER= data set option, BL_SFTP_WAIT_MILLISECONDS= data set option, BULKLOAD= data set option, FILENAME statement for the SFTP access method in SAS Statements: Reference

Syntax

BL_SFTP_HOST='host'

Details

You can specify the name of the host or the IP address of the computer. In most cases, this should be the server name of the SAP HANA server. If you do not set this option, the host name for SFTP defaults to the SAP HANA server name.

BL_SFTP_OPTIONS= Data Set Option

Specifies additional configuration options for Secure File Transfer Protocol (SFTP) access, such as port numbers.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Access
Alias: BL_SFTP_OPTIONS
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: SAP HANA

See: BL_SFTP_HOST= data set option, BL_SFTP_USER= data set option, BL_SFTP_WAIT_MILLISECONDS= data set option, BULKLOAD= data set option, FILENAME statement for the SFTP access method in SAS Statements: Reference

Syntax

BL_SFTP_OPTIONS=sftp-options

BL_SFTP_USER= Data Set Option

Specifies the user name.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Access
Alias: BL_SFTP_USER
Default: none
Requirement: To specify this option, you must first set BULKLOAD=YES.
Data source: SAP HANA
BL_SFTP_WAIT_MILLISECONDS= Data Set Option

Specifies the Secure File Transfer Protocol (SFTP) response time in milliseconds.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Access

Aliases: BL_SFTP_WAIT_MILLISECONDS

BL_WAIT_MILLISECONDS

Default: none

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: SAP HANA

See: BL_SFTP_HOST= data set option, BL_SFTP_OPTIONS= data set option, BL_SFTP_USER= data set option, BULKLOAD= data set option, FILENAME statement for the SFTP access method in SAS Statements: Reference

Syntax

BL_SFTP_USER='user'

BL_SFTP_WAIT_MILLISECONDS=option-string

BL_SQLLDR_PATH= Data Set Option

Specifies the location of the SQLLDR executable file.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: SQLLDR

Restriction: This option is ignored on z/OS.

Requirement: To specify this option, you must first set BULKLOAD=YES.

Data source: Oracle

Tip: Normally there is no need to specify this option for Oracle because the environment is set up to automatically find the QL*Loader.

See: BULKLOAD= data set option

Syntax

BL_SQLLDR_PATH=pathname
Syntax Description

pathname

specifies the full pathname to the Oracle SQLLDR executable file so that the engine SAS/ACCESS engine can call the Oracle SQL*Loader.

BL_SUPPRESS_NULLIF= Data Set Option

Indicates whether to suppress the NULLIF clause for the specified columns to increase performance when a table is created.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Requirement: To specify this option, you must first set BULKLOAD=YES. If you specify more than one column name, you must separate the names with spaces.
Data source: Oracle
See: BULKLOAD= data set option

Syntax

BL_SUPPRESS_NULLIF=(_ALL_=YES | NO)
BL_SUPPRESS_NULLIF=(column-name-1=YES | NO)< ... column-name-N=YES | NO>)

Syntax Description

column-name-N=YES | NO

specifies whether the NULLIF clause should be suppressed for the specified column in the table.

_ALL_=YES | NO

specifies whether the NULLIF clause should be suppressed for all columns.

Details

This option processes values from left to right. If you specify a column name twice or use the _ALL_ value, the last value overrides the first value that you specified for the column.

CAUTION:

If you set this option to YES and try to insert null values, unpredictable values are inserted into the column.

Examples

Example 1: Suppress NULLIF for Specific Table Columns

In this example, BL_SUPPRESS_NULLIF= in the DATA step suppresses the NULLIF clause for columns C1 and C5 in the table.

data x.suppressnullif2_yes (bulkload=yes BL_SUPPRESS_NULLIF=(c1=yes c5=yes));
run;
Example 2: Suppress NULLIF for All Table Columns

In this example, BL_SUPPRESS_NULLIF= in the DATA step suppresses the NULLIF clause for all columns in the table.

```sas
libname x oracle user=myusr1 pw=mypwd1 path=mypath;
%let num=1000000;   /* 1 million rows */
data x.testlmn ( bulkload=yes
   BL_SUPPRESS_NULLIF=( _all_ =yes )
   rename=(year=yearx) );
set x.big1mil (obs= &num ) ;
run;
```

BL_USE_PIPE= Data Set Option

Specifies whether to use a named pipe for data transfer.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaults:</td>
<td>YES (Netezza, Sybase IQ)</td>
</tr>
<tr>
<td></td>
<td>NO (Oracle, Greenplum, HAWQ)</td>
</tr>
<tr>
<td>Restrictions:</td>
<td>Not available on z/OS [Oracle]</td>
</tr>
<tr>
<td></td>
<td>Applies to UNIX environments only [Greenplum, HAWQ]</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES [Greenplum, HAWQ, Netezza, Oracle, Sybase IQ], or BULKUNLOAD=YES [Netezza, Oracle].</td>
</tr>
<tr>
<td>Data source:</td>
<td>Greenplum, HAWQ, Netezza, Oracle, Sybase IQ</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for Greenplum and HAWQ was added in the third maintenance release of SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>BL_DATAFILE= data set option, BULKLOAD= data set option, BULKUNLOAD= LIBNAME option, BULKUNLOAD= data set option</td>
</tr>
</tbody>
</table>

Syntax

`BL_USE_PIPE= YES | NO`

Syntax Description

**YES**

specifies that a named pipe is used to transfer data between SAS/ACCESS interfaces and the DBMS client interface.

**NO**

specifies that a flat file is used to transfer data.

Details

By default, the DBMS interface uses a named pipe interface to transfer large amounts of data between SAS and the DBMS when using bulk loading or unloading. If you prefer to use a flat data file that you can save for later use or examination, specify `BL_USE_PIPE=NO`. 
Example: Sample Code That Uses BL_USE_PIPE=YES

```sql
proc sql;
create table mydblib.mileages 
  (BULKLOAD=YES
   BL_USE_PIPE=YES
   BL_HOST='192.168.x.x'
   BL_PORT=8081)
  as select * from sashelp.mileages;
quit;
```

---

**BL_WARNING_COUNT= Data Set Option**

Specifies the maximum number of row warnings to allow before the load fails.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>2147483646</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To specify this option, you must first set BULKLOAD=YES and also specify a value for BL_REMOTE_FILE=.</td>
</tr>
<tr>
<td>Data source:</td>
<td>DB2 under UNIX and PC Hosts</td>
</tr>
<tr>
<td>See:</td>
<td>BL_REMOTE_FILE= data set option, BULKLOAD= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

```
BL_WARNING_COUNT=warning-count
```

**Syntax Description**

`warning-count`

specifies the maximum number of row warnings to allow before the load fails.

**Details**

Use this option to limit the maximum number of rows that generate warnings. See the log file for information about why the rows generated warnings.

---

**BUFFERS= Data Set Option**

Specifies the number of shared memory buffers to use for transferring data from SAS to Teradata.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>2</td>
</tr>
<tr>
<td>Data source:</td>
<td>Teradata</td>
</tr>
<tr>
<td>See:</td>
<td>MBUFFSIZE= data set option, MULTILOAD= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

```
BUFFERS=number-of-shared-memory-buffers
```
**Syntax Description**

**number-of-shared-memory-buffers**

a numeric value between 1 and 8 that specifies the number of buffers used for transferring data from SAS to Teradata.

**Details**

BUFFERS= specifies the number of data buffers to use for transferring data from SAS to Teradata. When you use the **MULTILOAD= data set option**, data is transferred from SAS to Teradata using shared memory segments. The default shared memory buffer size is 64K. The default number of shared memory buffers used for the transfer is 2.

Use BUFFERS= to vary the number of buffers for data transfer from 1 to 8. Specify the **MBUFSIZE= data set option** to vary the size of the shared memory buffers from the size of each data row up to 1MB.

---

**BULK_BUFFER= Data Set Option**

Specifies the number of bulk rows that the SAS/ACCESS engine can buffer for output.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** 100
- **Data source:** Sybase
- **See:** BULKLOAD= data set option, ENABLE_BULK= LIBNAME option

**Syntax**

**BULK_BUFFER=numeric-value**

**Syntax Description**

**numeric-value**

specifies the maximum number of rows that are allowed. This value depends on the amount of memory that is available to your system.

**Details**

This option improves performance by specifying the number of rows that can be held in memory for efficient retrieval from the DBMS. A higher number signifies that more rows can be held in memory and accessed quickly during output operations.

---

**BULKLOAD= Data Set Option**

Loads rows of data as one unit.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Aliases:** BL_DB2LDUTIL= [DB2 under z/OS]
  - BCP= [ODBC, OLE DB]
  - FASTLOAD= [Teradata]
- **Default:** NO
Before you set BULKLOAD=YES, you must have set the SAS_HADOOP_RESTFUL environment variable to 1 before starting your SAS session. For more information, see SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS.

**Data source:**
Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata

**Note:** Support for PostgreSQL was added for SAS 9.4. Support for Impala was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.

**Tip:** Using BULKLOAD=YES is the fastest way to insert rows into a DBMS table.

**See:** BULKUNLOAD= LIBNAME option, BULKUNLOAD= data set option, DBCOMMIT= data set option, ENABLE_BULK= LIBNAME option [Sybase], ERRLIMIT= data set option, bulk-load details in the DBMS-specific reference section for your SAS/ACCESS interface

### Syntax

**BULKLOAD** = **YES | NO**

**Syntax Description**

**YES**
calls a DBMS-specific bulk-load facility to insert or append rows to a DBMS table.

**NO**
uses the dynamic SAS/ACCESS engine to insert or append data to a DBMS table.

### BULKUNLOAD= Data Set Option

Rapidly retrieves (fetches) large number of rows from a data set.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Defaults:** NO [Netezza]
LIBNAME option value [Aster]

**Restriction:** To set BULKUNLOAD=YES in Netezza, you must be the Netezza admin user, or you must have Create External Table permission.

**Data source:** Aster, Netezza

**Note:** Support for Aster was added in the second maintenance release for SAS 9.4.

**Tip:** Using BULKLOAD=YES is the fastest way to retrieve large numbers of rows from a table.

**See:** BULKUNLOAD= LIBNAME option, BL_DATAFILE= data set option, BL_DELIMITER= data set option, BL_DELIMITER= data set option, BL_USE_PIPE= data set option, BULKLOAD= data set option, bulk loading on page 497 and unloading on page 498 for Aster, bulk unloading for Netezza

### Syntax

**BULKUNLOAD** = **YES | NO**
Syntax Description

YES
Netezza: calls the Netezza Remote External Table interface to retrieve data from the Performance Server.
Aster: uses the Aster client tool, ncluster_export, to retrieve data from the Aster DBMS.

NO
uses standard result sets to retrieve data from the DBMS.

BUSINESS_DATATYPE= Data Set Option

specifies the data type of the BUS_START and BUS_END columns in tables that contain business time (temporal) data.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: TIMESTAMP(6)
Requirement: DB2 for z/OS
Data source: DB2 for z/OS
See: TEMPORAL= data set option, BUSINESS_TIMEFRAME= data set option, SYSTEM_TIMEFRAME= data set option, OVERLAPS= data set option

Syntax

BUSINESS_DATATYPE=DATE | TIMESTAMP(6)

Syntax Description

DATE
specifies that the business data type is a date.

TIMESTAMP(6)
specifies that the business data type is TIMESTAMP(6).

BUSINESS_TIMEFRAME= Data Set Option

lets you provide a date range or a datetime range to use when querying or modifying a temporal table.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirement: DB2 z/OS with SAS 9.4
Data source: DB2 for z/OS
See: TEMPORAL= data set option, SYSTEM_TIMEFRAME= data set option, OVERLAPS= data set option, BUSINESS_DATATYPE= data set option
Syntax

BUSINESS_TIMEFRAME = FROM date-or-datetime1 TO date-or-datetime2

Syntax Description

date-or-datetimeN
specifies the beginning or end of a time period that is used when you query or modify a table that contains temporal data. Use either a date or datetime value, depending on the value of the BUSINESS_DATATYPE= data set option.

CAST= Data Set Option

Specifies whether SAS or the Teradata DBMS server should perform data conversions.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Restriction: CAST= applies only when you are reading Teradata tables into SAS, not when you are writing Teradata tables from SAS. It also applies only to SQL that SAS generates for you. If you supply your own SQL with the explicit SQL feature of PROC SQL, you must code your own casting clauses to force data conversions in Teradata instead of SAS.

Data source: Teradata
See: CAST= LIBNAME option, CAST_OVERHEAD_MAXPERCENT= LIBNAME option, CAST_OVERHEAD_MAXPERCENT= data set option

Syntax

CAST= YES | NO

Syntax Description

YES
forces data conversions (casting) to be done on the Teradata DBMS server and overrides any data overhead percentage limit.

NO
forces data conversions to be done by SAS and overrides any data overhead percentage limit.

Details

Internally, SAS numbers and dates are floating-point values. Teradata has several formats for numbers, including integers, floating-point values, and decimal values. Number conversion must occur when you are reading Teradata numbers that are not floating points (Teradata FLOAT). SAS/ACCESS can use the Teradata CAST= function to cause Teradata to perform numeric conversions. The parallelism of Teradata makes it suitable for performing this work, particularly if you are running SAS on z/OS, where CPU activity can be costly.

CAST= can cause more data to be transferred from Teradata to SAS, as a result of the option forcing the Teradata type into a larger SAS type. For example, the CAST=
transfer of a Teradata BYTEINT to SAS floating point adds seven overhead bytes to each row transferred.

These Teradata types are candidates for casting:

- INTEGER
- BYTEINT
- SMALLINT
- DECIMAL
- DATE

SAS/ACCESS limits data expansion for CAST= to 20% to trade rapid data conversion by Teradata for extra data transmission. If casting does not exceed a 20% data increase, all candidate columns are cast. If the increase exceeds this limit, SAS attempts to cast Teradata DECIMAL types only. If casting only DECIMAL types still exceeds the increase limit, data conversions are done by SAS.

You can alter the casting rules by using either CAST= or CAST_OVERHEAD_MAXPERCENT= LIBNAME option. With CAST_OVERHEAD_MAXPERCENT=, you can change the 20% overhead limit. With CAST=, you can override the percentage rules:

- CAST=YES forces Teradata to cast all candidate columns.
- CAST=NO cancels all Teradata casting.

---

**CAST_OVERHEAD_MAXPERCENT= Data Set Option**

Specifies the overhead limit for data conversions to perform in Teradata instead of SAS.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** 20

**Data source:** Teradata

**See:** CAST= LIBNAME option, CAST= data set option, CAST_OVERHEAD_MAXPERCENT= LIBNAME option

**Syntax**

```
CAST_OVERHEAD_MAXPERCENT=\<n>\n```

**Syntax Description**

\<n> specifies any positive numeric value. The engine default is 20.

**Details**

Teradata INTEGER, BYTEINT, SMALLINT, and DATE columns require conversion when read in to SAS. Either Teradata or SAS can perform conversions. When Teradata performs the conversion, the row size that is transmitted to SAS using the Teradata CAST operator can increase. CAST_OVERHEAD_MAXPERCENT= limits the allowable increase, also called conversion overhead.
For more information about conversions, conversion overhead, and casting, see the CAST= LIBNAME option.

**Example: Increase the Allowable Overhead**

This example demonstrates the use of CAST_OVERHEAD_MAXPERCENT= to increase the allowable overhead to 40%.

```plaintext
proc print data=mydblib.emp (cast_overhead_maxpercent=40);
  where empno<1000;
run;
```

---

**CHAR_AS_BINARY= Data Set Option**

Specifies whether to fetch all character columns as binary values when reading data from DB2.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Set Control
- **Default:** NO
- **Supports:** NLS
- **Data source:** DB2 under UNIX and PC Hosts, DB2 under z/OS
  
  **Note:** Support for this data set option was added in the first maintenance release for SAS 9.4.

  **Tip:** Setting this option to YES applies a $HEXn format to all character variables. Data therefore is displayed in hexadecimal format.

---

**Syntax**

```
CHAR_AS_BINARY= YES | NO
```

---

**COLUMN_DELIMITER= Data Set Option**

Specifies the single character to use as a column (field) delimiter.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Aliases:** COL_DELIM=
  - COL_DELIMIT=
  - COL_DELIMITER=
- **Default:** '001 (Ctrl-A)
- **Restriction:** You must specify a single character or a three-digit decimal value. Other commonly used formats (for example, '  ', 0x09, or ' 09 'x are invalid.
- **Requirement:** Specify a single-character or three-digit decimal ASCII value between 001 and 127.
- **Data source:** Hadoop
- **Note:** Support for this data set option was added for SAS 9.4.
- **See:** ROW_DELIMITER= data set option
Examples:
COLUMN_DELIMITER='#'
COLUMN_DELIMITER=009 (tab)

Syntax
COLUMN_DELIMITER=single-character

COMMAND_TIMEOUT= Data Set Option
Specifies the number of seconds to wait before a command times out.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: LIBNAME option setting
Data source: OLE DB
See: COMMAND_TIMEOUT= LIBNAME option

Syntax
COMMAND_TIMEOUT=number-of-seconds

Syntax Description
number-of-seconds
an integer greater than or equal to 0, where 0 represents no time-out.

CONFIG= Data Set Option
Specifies a path to the XML configuration file to use for bulk-load options.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Alias: CFG=, HD_CONFIG=
Default: LIBNAME option setting
Requirement: You must enclose the value in quotation marks.
Data source: Hadoop, Impala
Note: Support for this option was added in the third maintenance release for SAS 9.4.
See: BULKLOAD= LIBNAME option, CONFIG= LIBNAME option, CONFIGDIR= LIBNAME option, CONFIGDIR= data set option

Syntax
CONFIG='config-file'
Required Argument

**config-file**

specifies the path for the required configuration file for bulk-loading options. This single XML file must contain information about the host, port, and Kerberos principal for HDFS.

---

**CONFIGDIR= Data Set Option**

Specifies a directory path to search for the required XML configuration files to use for bulk-load options.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS)
- **Alias:** CFGDIR=, HD_CONFIGDIR=
- **Default:** LIBNAME option setting
- **Requirement:** You must enclose the value in quotation marks.
- **Data source:** Hadoop, Impala
- **Note:** Support for this option was added in the third maintenance release for SAS 9.4.
- **See:** BULKLOAD= LIBNAME option, CONFIG= LIBNAME option, CONFIG= data set option, CONFIGDIR= LIBNAME option

**Syntax**

```
CONFIGDIR='config-dir'
```

**Required Argument**

**config-dir**

specifies the directory path to search for the required configuration files (core-site.xml and hdfs-site.xml) for bulk-loading options that contain the required host, port, and Kerberos principal for HDFS.

---

**CURSOR_TYPE= Data Set Option**

Specifies the cursor type for read only and updatable cursors.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** LIBNAME option setting
- **Data source:** DB2 under UNIX and PC Hosts, Impala, Microsoft SQL Server, ODBC, OLE DB, Sybase IQ
- **See:** COMMAND_TIMEOUT= LIBNAME option, CURSOR_TYPE= LIBNAME option, KEYSET_SIZE= data set option [only Microsoft SQL Server and ODBC]

**Syntax**

```
CURSOR_TYPE=DYNAMIC | FORWARD_ONLY | KEYSET_DRIVEN | STATIC
```
Syntax Description

**DYNAMIC**
specifies that the cursor reflects all changes that are made to the rows in a result set as you move the cursor. The data values and the membership of rows in the cursor can change dynamically on each fetch. This is the default for the DB2 under UNIX and PC Hosts, Microsoft SQL Server, and ODBC interfaces. For OLE DB details, see “Details.”

**FORWARD_ONLY [not valid for OLE DB]**
specifies that the cursor functions like a DYNAMIC cursor except that it supports only sequential fetching of rows.

**KEYSETDriven**
specifies that the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows are reflected as you move the cursor.

**STATIC**
specifies that the cursor builds the complete result set when the cursor is opened. No changes made to the rows in the result set after the cursor is opened are reflected in the cursor. Static cursors are read-only.

Details

Not all drivers support all cursor types. An error is returned if the specified cursor type is not supported. The driver is allowed to modify the default without an error. See your database documentation for more information.

When no options have been set yet, here are the initial DBMS-specific defaults.

<table>
<thead>
<tr>
<th>DB2 for UNIX and PC</th>
<th>Microsoft SQL Server</th>
<th>ODBC</th>
<th>OLE DB</th>
<th>Sybase IQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYSETDriven</td>
<td>DYNAMIC</td>
<td>FORWARDONLY</td>
<td>FORWARDONLY</td>
<td>DYNAMIC</td>
</tr>
</tbody>
</table>

Here are the operation-specific defaults.

<table>
<thead>
<tr>
<th>Operation</th>
<th>DB2 for UNIX and PC</th>
<th>Microsoft SQL Server</th>
<th>ODBC, Sybase IQ</th>
<th>OLE DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert (UPDATE_SQL=NO)</td>
<td>KEYSETDriven</td>
<td>DYNAMIC</td>
<td>KEYSETDriven</td>
<td>FORWARDONLY</td>
</tr>
<tr>
<td>read (such as PROC PRINT)</td>
<td>driver default</td>
<td></td>
<td>driver default</td>
<td>(FORWARDONLY)</td>
</tr>
<tr>
<td>update (UPDATE_SQL=NO)</td>
<td>KEYSETDriven</td>
<td>DYNAMIC</td>
<td>KEYSETDriven</td>
<td>FORWARDONLY</td>
</tr>
<tr>
<td>CONNECTION=GLOBAL</td>
<td>DYNAMIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECTION=SHARED</td>
<td>DYNAMIC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OLE DB: Here are the OLE DB properties that are applied to an open rowset. For details, see your OLE DB programmer reference documentation.

<table>
<thead>
<tr>
<th>CURSOR_TYPE=</th>
<th>OLE DB Properties Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORWARD_ONLY or DYNAMIC (see “Details”)</td>
<td>DBPROP_OTHERINSERT=TRUE, DBPROP_OTHERUPDATEDELETE=TRUE</td>
</tr>
<tr>
<td>KEYSET_DRIVEN</td>
<td>DBPROP_OTHERINSERT=FALSE, DBPROP_OTHERUPDATEDELETE=TRUE</td>
</tr>
<tr>
<td>STATIC</td>
<td>DBPROP_OTHERINSERT=FALSE, DBPROP_OTHERUPDATEDELETE=FALSE</td>
</tr>
</tbody>
</table>

**DATETIME2= Data Set Option**

Specifies the scale for the timestamp literal for Microsoft SQL Server 2008 and the native Microsoft driver.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Data source:** ODBC
- **See:** DATETIME2= LIBNAME option

**Syntax**

DATETIME2= **YES | NO**

**Syntax Description**

**YES**

specifies a DATETIME precision and scale when creating the timestamp for the WHERE clause.

**NO**

uses the first occurring DATETIME precision and scale when creating the timestamp for the WHERE.

**DB_ONE_CONNECT_PER_THREAD= Data Set Option**

Specifies whether to limit the number of connections to the DBMS server for a threaded Read.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** YES
- **Data source:** Oracle
- **See:** Autopartitioning Scheme for Oracle
Syntax

DB_ONE_CONNECT_PER_THREAD= YES | NO

Syntax Description

YES
  enables this option, allowing only one connection per partition.

NO
  disables this option.

Details

Use this option if you want to have only one connection per partition. By default, the number of connections is limited to the maximum number of allowed threads. If the value of the maximum number of allowed threads is less than the number of partitions on the table, a single connection reads multiple partitions.

DBCOMMIT= Data Set Option

Causes an automatic COMMIT (a permanent writing of data to the DBMS) after a specified number of rows are processed.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Alias: CHECKPOINT= [Teradata]

Default: LIBNAME option setting

Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: BULKLOAD= LIBNAME option, BULKLOAD= data set option, DBCOMMIT= LIBNAME option, ERRLIMIT= LIBNAME option, ERRLIMIT= data set option, INSERT_SQL= LIBNAME option, INSERT_SQL= data set option, INSERTBUFF= LIBNAME option, INSERTBUFF= data set option, ML_CHECKPOINT= data set option, "Using FastLoad"

Syntax

DBCOMMIT= n

Syntax Description

n
  specifies an integer greater than or equal to 0.

Details

DBCOMMIT= affects update, delete, and insert processing. The number of rows processed includes rows that are not processed successfully. When DBCOMMIT=0, COMMIT is issued only once—after the procedure or DATA step completes.
If you explicitly set the DBCOMMIT= option, SAS/ACCESS fails any update with a WHERE clause.

If you specify both DBCOMMIT= and ERRLIMIT= and these options collide during processing, COMMIT is issued first and ROLLBACK is issued second. Because COMMIT is issued (through the DBCOMMIT= option) before ROLLBACK (through the ERRLIMIT= option), DBCOMMIT= overrides ERRLIMIT=.

**DB2 Under UNIX and PC Hosts:** When BULKLOAD=YES, the default is 10000.

**Teradata:** For the default behavior of this option, see the FastLoad description in the Teradata section. DBCOMMIT= is disabled for MultiLoad to prevent any conflict with ML_CHECKPOINT=.

### Example: Specify the Number of Row to Process

A commit is issued after every 10 rows are processed in this example:

```sas
data oracle.dept(dbcommit=10);
  set myoralib.staff;
run;
```

---

**DBCONDITION= Data Set Option**

Specifies criteria for subsetting and ordering DBMS data.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** none

**Restrictions:** The DBKEY= and DBINDEX= options are ignored when you use DBCONDITION=.

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** DBINDEX= data set option, DBKEY= data set option

---

**Syntax**

```
DBCONDITION="DBMS-SQL-query-clause"
```

**Syntax Description**

**DBMS-SQL-query-clause** specifies a DBMS-specific SQL query clause, such as WHERE, GROUP BY, HAVING, or ORDER BY.

**Details**

You can use this option to specify selection criteria in the form of DBMS-specific SQL query clauses, which the SAS/ACCESS engine passes directly to the DBMS for processing. When selection criteria are passed directly to the DBMS for processing,
performance is often enhanced. The DBMS checks the criteria for syntax errors when it receives the SQL query.

**Example: Return Only Condition-Specific Rows**

In this example, the function that is passed to the DBMS with the DBCONDITION= option causes the DBMS to return to SAS only those rows that satisfy the condition.

```sql
proc sql;
  create view smithnames as
  select lastname from myoralib.employees
    (dbcondition="where soundex(lastname) = soundex('SMYTHE')")
  using libname myoralib oracle user=myusr1
    pw=mypwd1 path=mysrv1;
select lastname from smithnames;
```

———

**DBCONSTRAINT= Data Set Option**

Provides table-level definitions to specify when a table is created.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Requirement:** You must specify constraints within the CREATE TABLE statement parentheses.
- **Data source:** Teradata
- **See:** DBCREATE_TABLE_OPTS= data set option

**Syntax**

`DBCONSTRAINT='DBMS-SQL-clauses'`

**Syntax Description**

- **DBMS-SQL-clauses** indicates one or more clauses that are specific to Teradata that must be specified when creating a table but that must appear inside the CREATE TABLE parentheses.

**Details**

Use this option to add table-level definitions in the CREATE TABLE statement. DBCREATE_TABLE_OPTS= is similar to this option except that it lets you add DBMS-specific text outside (to the right) of the parentheses.

**Example: Specify Primary Key Columns for a Table**

In this example, DBCONSTRAINT= specifies a table-level constraint that columns x and y are primary key columns.

```sql
libname x teradata user=myusr1 pw=mypwd1;

iscrim case=lower dbmsoptions='primary key x, y'
    create view myview as
    select y, x from mytable;
```

/*
 * Submits this SQL with table-level constraints.
 */
CREATE MULTISET TABLE "test"
  ("x" FLOAT NOT NULL ,
  "y" FLOAT NOT NULL ,
  CONSTRAINT test PRIMARY KEY(X,Y)
 );

data x.test(DBCONSTRAINT='CONSTRAINT test PRIMARY KEY(X,Y)' DBNULL=(_ALL_=NO));
x=1;y=1;
run;

DBCREATE_TABLE_EXTERNAL= Data Set Option

Specifies the type of table to create and how associated data files are handled.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Aliases: DBCREATE_EXTERNAL=

Default: NO

Interaction: You can specify this option, the DBCREATE_TABLE_LOCATION= option, or both.

Data source: Hadoop

Tip: This option determines only the disposition of a file upon delete.

See: DBCREATE_TABLE_EXTERNAL= LIBNAME option,
    DBCREATE_TABLE_LOCATION= data set option, DBCREATE_TABLE_OPTS= LIBNAME option, DBCREATE_TABLE_OPTS= data set option

Syntax

DBCREATE_TABLE_EXTERNAL=YES | NO

Syntax Description

YES
  creates an external table—one that is stored outside of the Hive warehouse.

NO
  creates a managed table—one that is managed within the Hive warehouse.

Details

When a managed table is dropped, its data is also deleted. When an external table is dropped, its data is preserved. Create an EXTERNAL table if you want to preserve table data if the table is dropped. SAS issues a DROP TABLE statement when PROC DELETE references a Hive table and also with the DROP TABLE statement in PROC SQL.

Example: Protect Data from DROP TABLE

In this example, DBCREATE_TABLE_LOCATION= stores the table data outside of the Hive warehouse. DBCREATE_TABLE_EXTERNAL=YES protects the data from being deleted if the table is dropped.
Identifies the HDFS location of the root directory for storing table data.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Aliases: DBCREATE_LOCATION=
         DBCREATE_LOC=
         DBCREATE_PATH=

Default: /user/hive/warehouse/tablename [with the default schema], /user/hive/warehouse/
         schema.db/tablename [with a nondefault schema]

Interaction: You can specify this option, the DBCREATE_TABLE_EXTERNAL= option, or both.

Data source: Hadoop

Tip: This option determines only the physical location of a file.

See: DBCREATE_TABLE_EXTERNAL= LIBNAME option,
     DBCREATE_TABLE_EXTERNAL= data set option, DBCREATE_TABLE_OPTS= LIBNAME option, DBCREATE_TABLE_OPTS= data set option

Syntax

DBCREATE_TABLE_LOCATION="path"

Syntax Description

'path'

specifies the HDFS location of the root directory for storing table data.

Details

Use this option to specify an alternative HDFS location, which adds the LOCATION keyword to the CREATE TABLE DDL.

Example: Creating a File in an Alternative Hive Depository

Both DBCREATE_TABLE_EXTERNAL= and DBCREATE_TABLE_LOCATION= options are set in this example.

    LIBNAME db HADOOP SERVER=mysrv1 USER=myusr1 DB=myschema1;
    DATA db.mytab {
        DBCREATE_TABLE_EXTERNAL=YES
        DBCREATE_TABLE_LOCATION="/mydir/mytab";
        SET mydata;
        RUN;
    }
**DBCREATE_TABLE_OPTS= Data Set Option**

Specifies DBMS-specific syntax to add to the end of the CREATE TABLE statement.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Alias:** POST_STMT_OPTS=

**Default:** LIBNAME option setting

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase Iq, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**Tips:**
- If you are already using DBTYPE= within an SQL CREATE TABLE statement, you can also use it to include column modifiers.
- If you want all output tables to be in the default (non-TEXTFILE) format, use the LIBNAME option. (See the LIBNAME option for examples.)

**See:** DBCREATE_TABLE_EXTERNAL= LIBNAME option, DBCREATE_TABLE_EXTERNAL= data set option, DBCREATE_TABLE_LOCATION= data set option, DBCREATE_TABLE_OPTS= LIBNAME option, DBTYPE= data set option, POST_STMT_OPTS= data set option, POST_TABLE_OPTS= data set option, PRE_STMT_OPTS= data set option, PRE_TABLE_OPTS= data set option

**Syntax**

```
DBCREATE_TABLE_OPTS='DBMS-SQL-clauses'
```

**Required Argument**

**DBMS-SQL-clauses** specifies one or more DBMS-specific clauses that can be appended to the end of an SQL CREATE TABLE statement.

**Details**

You can use this option to add DBMS-specific clauses at the end of the SQL CREATE TABLE statement. The SAS/ACCESS engine passes the SQL CREATE TABLE statement and its clauses to the DBMS. The DBMS then executes the statement and creates the DBMS table. This option applies only when you are creating a DBMS table by specifying a libref associated with DBMS data.

If you need to add an option in a location other than at the end of your CREATE TABLE statement, use one of these data set options: POST_TABLE_OPTS=, PRE_STMT_OPTS=, and PRE_TABLE_OPTS=. For example, for Greenplum, a WITH clause should appear after the table name but before a DISTRIBUTED RANDOMLY clause in a CREATE TABLE statement. You should therefore specify a WITH clause using the POST_TABLE_OPTS= data set option.
Examples

**Example 1: Partition a DB2 Table**
In this example, the DB2 table TEMP is created with the value of the DBCREATE_TABLE_OPTS= option appended to the CREATE TABLE statement.

```sas
libname mydblib db2 user=myusr1 pwd=mypwd1 dsn=sample;
data mydblib.temp (DBCREATE_TABLE_OPTS='PARTITIONING KEY (X) USING HASHING');
x=1; output;
x=2; output;
run;
```

When you use this data set option to create the DB2 table, the SAS/ACCESS Interface to DB2 passes this DB2 SQL statement:

```
CREATE TABLE TEMP (X DOUBLE) PARTITIONING KEY (X) USING HASHING
```

**Example 2: Partition a Hive Table**
In this example, a Hive table PART is created with the value of the DBCREATE_TABLE_OPTS= option appended to the CREATE TABLE statement.

```sas
%dbtrace(",,,d");
libname x HADOOP server=XXXX user=XXXXX pwd=XXXXXX ;
data x.part (DBCREATE_TABLE_OPTS="PARTITIONED BY(I INT)");
i=1; output;
j=2; output;
run;
```

When you use this data set option to create this table, the Hadoop interface generates a CREATE TABLE statement similar to this one.

```
HADOOP_8: Executed: on connection 2
CREATE TABLE `PART` (`j` DOUBLE) PARTITIONED BY(I INT)
TBLPROPERTIES ("SAS OS Name"='W32_7PRO', 'SAS Version'='9.04.01M3D04152015')
```

---

**DBFORCE= Data Set Option**

Specifies whether to force data truncation during insert processing.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>NO</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>DBTYPE= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

```
DBFORCE= YES | NO
```
Syntax Description

YES
specifies that rows that contain data values that exceed the length of the DBMS column are inserted, and the data values are truncated to fit the DBMS column length.

NO
specifies that the rows that contain data values that exceed the DBMS column length are not inserted.

Details

This option determines how the SAS/ACCESS engine handles rows that contain data values that exceed the length of the DBMS column. DBFORCE= works only when you create a DBMS table with DBTYPE= data set option—namely, you must specify both DBFORCE= and DBTYPE=. DBFORCE= does not work for inserts or updates. Therefore, to insert or update a DBMS table, you cannot use the DBFORCE= option—you must instead specify the options that are available with SAS procedures. For example, specify the FORCE= data set option in SAS with PROC APPEND.

FORCE= overrides DBFORCE= when you use FORCE= with PROC APPEND or the PROC SQL UPDATE statement. PROC SQL UPDATE does not warn you before it truncates data.

Oracle: You must set DBFORCE=YES if you use DBTYPE= to override the default data type of VARCHAR2 with NVARCHAR2 or NCHAR.

Example: Truncate Data during Insert Processing

In this example, two librefs are associated with Oracle databases, and it does not specify databases and schemas because it uses the defaults. In the DATA step, MYDBLIB.DEPT is created from the Oracle data that MYORALIB.STAFF references. The LASTNAME variable is a character variable of length 20 in MYORALIB.STAFF. When MYDBLIB.DEPT is created, the LASTNAME variable is stored as a column of type character and length 10 by using DBFORCE=YES.

libname myoralib oracle user=tester1 password=tst1;
libname mydblib oracle user=myusr1 password=mypwd1;
data mydblib.dept(dbtype=(lastname='char(10)')
dbforce=yes);
set myoralib.staff;
run;

DBGEN_NAME= Data Set Option

Specifies how SAS automatically renames columns (when they contain characters that SAS does not allow, such as $) to valid SAS variable names.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>DBMS</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
</tbody>
</table>
See:  DBGEN_NAME= LIBNAME option, VALIDVARNAME= system option

Syntax

DBGEN_NAME=DBMS | SAS

Syntax Description

DBMS
specifies that SAS renames DBMS columns to valid SAS variable names. SAS converts any disallowed characters to underscores. If it converts a column to a name that already exists, it appends a sequence number at the end of the new name.

SAS
specifies that SAS converts DBMS columns with disallowed characters into valid SAS variable names. SAS uses the format _COLn, where n is the column number, starting with 0. If SAS converts a name to a name that already exists, it appends a sequence number at the end of the new name.

Details

SAS retains column names when it reads data from DBMS tables unless a column name contains characters that SAS does not allow, such as $ or %. SAS allows alphanumeric characters and the underscore (_).

This option is intended primarily for National Language Support, notably converting kanji to English characters. English characters that are converted from kanji are often those that SAS does not allow. Although this option works for the single-byte character set (SBCS) version of SAS, SAS ignores it in the double-byte character set (DBCS) version. So if you have the DBCS version, you must first set VALIDVARNAME=ANY before using your language characters as column variables.

Example

If you specify DBGEN_NAME=SAS, SAS renames a DBMS column named Dept$Amt to _COLn. If you specify DBGEN_NAME=DBMS, SAS renames the Dept$Amt column to Dept_Amt.

DBINDEX= Data Set Option

Detects and verifies that indexes exist on a DBMS table.

Valid in:  DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Defaults:  YES (Aster, DB2 under UNIX and PC Hosts, Greenplum, Microsoft SQL Server, Netezza, ODBC, PostgreSQL, Sybase IQ, Vertica)
NO (Informix, MySQL, OLE DB, Oracle, SAP HANA, Sybase, Teradata)

Note:  Support for HAWQ was added in the third maintenance release for SAS 9.4.

See:  DBINDEX= LIBNAME option, DBKEY= data set option, MULTI_DATASRC_OPT= LIBNAME option

CAUTION:  Improper use of this option can impair performance. See “Using the DBINDEX=, DBKEY=, and MULTI_DATASRC_OPT= Options” on page 50 for detailed information about using this option.
Syntax

DBINDEX= YES | NO | <index-name>

Syntax Description

YES
triggers the SAS/ACCESS engine to search for all indexes on a table and return them to SAS for evaluation. If SAS/ACCESS finds a usable index, it passes the join WHERE clause to the DBMS for processing. A usable index should have at least the same attributes as the join column.

NO
indicates that no automated index search is performed.

index-name
verifies the index name that is specified for the index columns on the DBMS table. It requires the same type of call as when DBINDEX=YES is used.

Details

If indexes exist on a DBMS table and are of the correct type, you can use this option to potentially improve performance when you are processing a join query. Performance is often improved for queries that involve a large DBMS table and a relatively small SAS data set that is passed to the DBMS.

Queries must be issued to the necessary DBMS control or system tables to extract index information about a specific table or validate the index that you specified.

You can enter the DBINDEX= option as a LIBNAME option, SAS data set option, or an option with PROC SQL. Here is the order in which the engine processes it:

1. DATA step or PROC SQL specification.
2. LIBNAME statement specification

Specifying DBKEY= takes precedence over DBINDEX=.

Examples

Example 1
Here is the SAS data set that is used in these examples.

data s1;
  a=1; y='aaaaa'; output;
  a=2; y='bbbb'; output;
  a=5; y='cccc'; output;
run;

Example 2: Use DBINDEX= in a LIBNAME Statement

libname mydblib oracle user=myuser password=userpwd dbindex=yes;
proc sql;
select * from s1 aa, x.dbtab bb where aa.a=bb.a;
select * from s1 aa, mydblib.dbtab bb where aa.a=bb.a;
The DBINDEX= values for table Dbtab are retrieved from the DBMS and compared with the join values. In this case, a match was found so that the join is passed down to the DBMS using the index. If the index was not found, the join would take place in SAS.

**Example 3: Use DBINDEX= in a SAS DATA Step**

```
data a;
set s1;
set x.dbtab(dbindex=yes) key=a;
set mydblib.dbtab(dbindex=yes) key=a;
run;
```

The key is validated against the list from the DBMS. If an index, a pass-down occurs. Otherwise, the join takes place in SAS.

**Example 4: Use DBINDEX= in PROC SQL**

```
proc sql;
select * from s1 aa, x.dbtab(dbindex=yes) bb where aa.a=bb.a;
select * from s1 aa, mylib.dbtab(dbindex=yes) bb where aa.a=bb.a;
/*or*/
select * from s1 aa, x.dbtab(dbindex=a) bb where aa.a=bb.a;
select * from s1 aa, mylib.dbtab(dbindex=a) bb where aa.a=bb.a;
```

---

**DBKEY= Data Set Option**

Specifies a key column to optimize DBMS retrieval.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** DBINDEX= data set option, DBNULLKEYS= data set option

**CAUTION:** Improper use of this option can decrease performance. For detailed information about using this option, see the DBINDEX= LIBNAME option.

**Syntax**

```
DBKEY=(<column-1> <… <column-n> > )
```

**Syntax Description**

- **column**

  SAS uses this to build an internal WHERE clause to search for matches in the DBMS table based on the key column. For example:

  ```
  select * from sas.a, dbms.b(dbkey=x) where a.x=b.x;
  ```
In this example, DBKEY= specifies column x, which matches the key column that the WHERE clause designates. However, if the DBKEY= column does NOT match the key column in the WHERE clause, DBKEY= is not used.

Details

You can use this option to potentially improve performance when you are processing a join that involves a large DBMS table and a small SAS data set or DBMS table.

When you specify DBKEY=, it is strongly recommended that an index exists for the key column in the underlying DBMS table. Performance can be severely degraded without an index.

Examples

**Example 1: Using DBKEY= with MODIFY=**

This example uses DBKEY= with the MODIFY statement in a DATA step:

```sas
libname invty db2;
data invty.stock;
  set addinv;
  modify invty.stock(dbkey=partno) key=dbkey;
  INSTOCK=instock+nwstock;
  RECDATE=today();
  if _iorc_=0 then replace;
runc;
```

**Example 2: Using More Than One DBKEY= Value**

To use more than one value for DBKEY=, you must include the second value as a join on the WHERE clause. In the next example PROC SQL brings the entire DBMS table into SAS and then proceeds with processing:

```sas
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc sql;
create table work.barbkey as
  select keyvalues.empid, employees.hiredate, employees.jobcode
  from mydblib.employees(dbkey=(empid jobcode))
    inner join work.keyvalues on employees.empid = keyvalues.empid;
quit;
```

**DBLABEL= Data Set Option**

Specifies whether to use SAS variable labels or SAS variable names as the DBMS column names during output processing.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Restriction:** This option is valid only for creating DBMS tables.
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Impala, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
Syntax

DBLABEL=YES | NO

Syntax Description

YES
specifies that SAS variable *labels* are used as DBMS column names during output processing.

NO
specifies that SAS variable *names* are used as DBMS column names.

Example: Specify a Variable Label

In this example, a SAS data set, NEW, is created with one variable C1. This variable is assigned a label of DEPTNUM. In the second DATA step, the MYDBLIB.MYDEPT table is created by using DEPTNUM as the DBMS column name. By setting DLBLABEL=YES, the label can be used as the column name.

data new;
  label c1='deptnum';
  c1=001;
run;
data mydblib.mydept(dylabel=yes);
  set new;
run;
proc print data=mydblib.mydept;
run;

DBLINK= Data Set Option

Specifies a link from your local database to database objects on another server [Oracle]. Specifies a link from your default database to another database on the server to which you are connected [Sybase].

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: LIBNAME option setting
Data source: Oracle, Sybase
See: DBLINK= LIBNAME option

Syntax

DBLINK=database-link

Details

This option operates differently in each DBMS.

Oracle: A link is a database object that identifies an object that is stored in a remote database. A link contains stored path information and can also contain user name and password information for connecting to the remote database. If you specify a link, SAS
uses the link to access remote objects. If you omit DBLINK=, SAS accesses objects in the local database.

*Sybase*: You can use this option to link to another database within the same server to which you are connected. If you omit DBLINK=, SAS can access objects only in your default database.

**Example: Specify an Oracle Link**

In this example, SAS sends MYORADB.EMPLOYEES to Oracle as EMPLOYEES@SALES.HQ.ACME.COM.

```sas
proc print data=myoradb.employees(dblink='sales.hq.acme.com');
run;
```

---

**DBMASTER= Data Set Option**

Designates which table is the larger table when you are processing a join that involves tables from two different types of databases.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** `MULTI_DATASRC_OPT= LIBNAME option`

**Syntax**

`DBMASTER=YES`

**Syntax Description**

**YES**

designates which of two tables that are referenced in a join operation is the larger table.

**Details**

You can use this option to specify which table reference in a join is the larger table. This can improve performance by eliminating the processing that is normally performed to determine this information. However, this option is ignored when outer joins are processed.

**Example: Join Two Tables**

In this example, a table from an Oracle database and a table from a DB2 database are joined. `DBMASTER=` is set to `YES` to indicate that the Oracle table is the larger table. The DB2 table is the smaller table.

```sas
libname mydblib oracle user=myusr1 /*database 1 */
```
pw=mypwd1 path='myorapath'
libname mydblib2 db2 user=myusr1 /*database 2 */
pw=mypwd1 path='mydb2path';
proc sql;
    select * from mydblib.bigtab(dbmaster=yes), mydblib2.smalltab
    bigtab.x=smalltab.x;

DBMAX_TEXT= Data Set Option

Determines the length of any very long DBMS character data type that is read into SAS or written from SAS when you are using a SAS/ACCESS engine.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Alias: TEXTSIZE [Vertica]
Default: 1024 [Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica], 32767 [Hadoop]
Restriction: This option applies to appending and updating rows in an existing table. It does not apply when creating a table.
Requirement: You must set the value to 4000 when you are using procedures that work with SAS High-Performance Analytics Server.
Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, Hadoop, HAWQ, Impala, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica
Notes: Support for HAWQ was added in the third maintenance release for SAS 9.4. If you set the value of DBMAX_TEXT= so that data in a table is truncated, the data load fails for that table. The number of bytes that are used to store characters might vary and is based on your session encoding.
Tip: DBMAX_TEXT= is usually used with a very long DBMS character data type, such as the Sybase TEXT data type or the Oracle CLOB data type.
See: DBMAX_TEXT= LIBNAME option

Syntax

DBMAX_TEXT=integer

Syntax Description

integer
    is a number between 1 and 32,767.

Details

Hadoop: This option applies for the STRING data type.

Oracle: For SAS 9 or higher, this option applies for CLOB, BLOB, LONG, LONG RAW, and LOB data types. The behavior of the ACCESS and DBLOAD procedures have not changed since SAS 8. So only LONG and LOB data types are valid if you use this option with those procedures. Also, this option is ignored if the data type is CHAR or VARCHAR2.
**DBNULL= Data Set Option**

Indicates whether NULL is a valid value for the specified columns when a table is created.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Defaults:** none (Greenplum, HAWQ, Teradata)
  - _ALL_=YES (DB2 under UNIX and PC Hosts, Impala, Informix, OLE DB, Sybase)
  - YES (Aster, DB2 under z/OS, Microsoft SQL Server, MySQL, Netezza, ODBC, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Vertica)
  - ALL_=NO (DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica)
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** NULLCHAR= data set option, NULLCHARVAL= data set option

**Syntax**

```
DBNULL=<_ALL_ | YES | NO >
    | (<column-name-1>=YES | NO> <...<column-name-n>=YES | NO> )
```

**Syntax Description**

- _ALL_ [valid only for Informix, Netezza, Oracle, Sybase, Teradata]
  - specifies that the YES or NO applies to all columns in the table.
- YES
  - specifies that the NULL value is valid for the specified columns in the DBMS table.
- NO
  - specifies that the NULL value is not valid for the specified columns in the DBMS table.

**Details**

This option is valid only for creating DBMS tables. If you specify more than one column name, you must separate them with spaces.

The DBNULL= option processes values from left to right. If you specify a column name twice or if you use the _ALL_ value, the last value overrides the first value that you specified for the column.

**Examples**

**Example 1: Prevent Specific Columns from Accepting Null Values**

In this example, you can use the DBNULL= option to prevent the EMPID and JOBCODE columns in the new MYDBLIB.MYDEPT2 table from accepting null values. If the EMPLOYEES table contains null values in the EMPID or JOBCODE columns, the DATA step fails.

```
data mydblib.mydept2(dbnull=(empid=no jobcode=no));
  set mydblib.employees;
```
Example 2: Prevent All Columns from Accepting Null Values

In this example, all columns in the new MYDBLIB.MYDEPT3 table except for the JOBCODE column are prevented from accepting null values. If the EMPLOYEES table contains null values in any column other than the JOBCODE column, the DATA step fails.

```
data mydblib.mydept3(dbnull=(_ALL_=no jobcode=YES));
  set mydblib.employees;
run;
```

**DBNULLKEYS= Data Set Option**

Controls the format of the WHERE clause with regard to NULL values when you use the DBKEY= data set option.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Vertica</td>
</tr>
<tr>
<td>Note:</td>
<td>Support for HAWQ was added in the third maintenance release for SAS 9.4.</td>
</tr>
<tr>
<td>See:</td>
<td>DBINDEX= data set option, DBKEY= data set option, DBNULLKEYS= LIBNAME option</td>
</tr>
</tbody>
</table>

**Syntax**

```
DBNULLKEYS= YES | NO
```

**Required Arguments**

YES

specifies that the key columns in a transaction table or a master table might contain NULL values.

NO

specifies that the key columns in a transaction table or a master table do not contain NULL values.

**Details**

If there might be NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, use DBNULLKEYS=YES. When you specify DBNULLKEYS=YES and specify a column that the DBKEY= data set option defines as NOT NULL, SAS generates a WHERE clause to find NULL values. For example, if you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause with this syntax:

```
WHERE ((COLUMN = ?) OR ((COLUMN IS NULL) AND (? IS NULL)))
```

This syntax enables SAS to prepare the statement once and use it for any value (NULL or NOT NULL) in the column. This syntax has the potential to be much less efficient
than the shorter form of the following WHERE clause. When you specify
DBNULLKEYS=NO or specify a column that is defined as NOT NULL in the
DBKEY= option, SAS generates a simple WHERE clause.

If you know that there are no NULL values in the transaction table or the master table
for the columns that you specify in the DBKEY= option, you can use
DBNULLKEYS=NO. If you specify DBNULLKEYS=NO and DBKEY=COLUMN,
SAS generates a shorter form of the WHERE clause, regardless of whether the column
that is specified in DBKEY= is defined as NOT NULL.

WHERE {COLUMN = ?}

---

**DBPROMPT= Data Set Option**

Specifies whether SAS displays a window that prompts you to enter DBMS connection information.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Defaults:** LibNAME option setting (Aster, DB2 under UNIX and PC Hosts, Greenplum, Impala,
Microsoft SQL Server, MySQL, Netezza, ODBC, PostgreSQL, SAP HANA, Sybase
IQ)

NO (Sybase, Vertica)

**Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Microsoft SQL Server,
MySQL, Netezza, ODBC, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: DBPROMPT= LIBNAME option

---

**Syntax**

DBPROMPT= YES | NO

**Syntax Description**

**YES**

displays the prompting window.

**NO**

does not display the prompting window.

---

**Details**

This data set option is supported only for view descriptors.

*Oracle:* You can enter 30 characters in the Oracle interface each for USERNAME and
PASSWORD and up to 70 characters for PATH, depending on your platform and
terminal type.

---

**Examples**

**Example 1: Use the Default Value (No Prompt)**

In this example, connection information is specified in the ACCESS procedure. The
DBPROMPT= data set option defaults to NO during the PRINT procedure because it is
not specified.
proc access dbms=oracle;
create a1ib.mydesc.access;
user=myusr1;
password=mypwd1;
table=dept;
create vlib.myview.view;
select all;
run;
proc print data=vlib.myview;
run;

Example 2: Prompt for Connection Information
In the next example, the DBPROMPT window appears during connection to the DBMS. Values that were previously specified during the creation of MYVIEW are pulled into the DBPROMPT window fields. You must edit or accept the connection information in the DBPROMPT window to proceed. The password value appears as a series of asterisks; you can edit it.

proc print data=vlib.myview(dbprompt=yes);
run;

DBSASLABEL= Data Set Option
Specifies how the engine returns column labels.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: COMPAT
Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: DBSASLABEL= LIBNAME option

Syntax

DBSASLABEL=COMPAT | NONE

Syntax Description

COMPAT
specifies that the labels returned should be compatible with what the application normally receives. In other words, engines exhibit their normal behavior.

NONE
specifies that the engine does not return a column label. The engine returns blanks for the column labels.

Details
By default, the SAS/ACCESS interface for your DBMS generates column labels from column names instead of from the real column labels.
You can use this option to override the default behavior. It is useful for when PROC SQL uses column labels as headings instead of column aliases.

Example: Return Blank Labels for Aliases in Headings

This example shows how to use DBSASLABEL= to return blank column labels so that PROC SQL can use the column aliases as the column headings.

```
proc sql;
  select deptno as Department ID, loc as Location
  from mylib.dept(dbsaslabel=none);
```

When DBSASLABEL=NONE, PROC SQL ignores the aliases, and it uses DEPTNO and LOC as column headings in the result set.

**DBSASTYPE= Data Set Option**

Specifies data types to override the default SAS data types during input processing.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** DBMS-specific
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**Syntax**

```
DBSASTYPE=(<column-name-1>=<SAS-data-type> <column-name-n>=<SAS-data-type> )
```

**Syntax Description**

- **column-name** specifies a DBMS column name.

- **SAS-data-type** specifies a SAS data type, which can be CHAR(n), NUMERIC, DATETIME, DATE, TIME. See the DBMS-specific reference section for your SAS/ACCESS interface for details.

**Details**

By default, the SAS/ACCESS interface for your DBMS converts each DBMS data type to a SAS data type during input processing. When you need a different data type, you can use this option to override the default and assign a SAS data type to each specified DBMS column. Some conversions might not be supported. In that case, SAS prints an error to the log.

If you convert a long string value to the NUMERIC type, the numeric value that is stored in SAS might not exactly match the original character value. This happens with long strings that contain more than 15 significant digits. For example, if SAS reads in a character value of '123456789012345678901234567890' and converts that to type NUMERIC, then the numeric value that SAS stores is
Examples

**Example 1: Override the Default Data Type**
In this example, DBSASTYPE= specifies a data type to use for the MYCOLUMN column when SAS prints ODBC data. SAS can print the values if the data in this DBMS column is stored in a format that SAS does not support, such as SQL_DOUBLE(20).

```sas
proc print data=mylib.mytable
   (dbsastype=(mycolumn='CHAR(20)'));
run;
```

**Example 2: Convert Column Length**
In the next example, data that is stored in the DBMS FIBERSIZE column has a data type that provides more precision than SAS can accurately support, such as DECIMAL(20). If you use only PROC PRINT on the DBMS table, the data might be rounded or are displayed as a missing value. So you could use DBSASTYPE= instead to convert the column so that the length of the character field is 21. The DBMS performs the conversion before the data is brought into SAS, so precision is preserved.

```sas
proc print data=mylib.specprod
   (dbsastype=(fibersize='CHAR(21)'));
run;
```

**Example 3: Append Tables to Match Data Types**
The next example uses DBSASTYPE= to append one table to another when the data types cannot be compared. If the EMPID variable in the SAS data set is defined as CHAR(20) and the EMPID column in the DBMS table is defined as DECIMAL(20), you can use DBSASTYPE= to make them match:

```sas
proc append base=dblib.hrdata (dbsastype=(empid='CHAR(20)'))
   data=saslib.personnel;
run;
```

DBSASTYPE= specifies to SAS that the EMPID is defined as a character field of length 20. When a row is inserted from the SAS data set into a DBMS table, the DBMS performs a conversion of the character field to the DBMS data type DECIMAL(20).

**DBSICE= Data Set Option**
Specifies user-supplied WHERE clauses to partition a DBMS query for threaded Reads.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** none

**Data source:** DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, ODBC, Oracle, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for Vertica was added for SAS 9.4. Support for SAP HANA was added in the first maintenance release for SAS 9.4. Support for Greenplum was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: DBSLICE= DBSLICE= LIBNAME option, DBSLICE= data set option

CAUTION: When using DBSLICE=, you are responsible for data integrity. If your WHERE clauses omit rows from the result set or retrieve the same row on more than one thread, your input DBMS result set is incorrect and your SAS program generates incorrect results.

Syntax

DBSLICE= ("WHERE-clause-1" "WHERE-clause-2" <..." WHERE-clause-n")

DBSLICE= (<server>= "WHERE-clause-1" <server>= "WHERE-clause-2" <... <server>= "WHERE-clause-n")

Syntax Description

WHERE-clause

The WHERE clauses in the syntax signify DBMS-valid WHERE clauses that partition the data. The clauses should not cause any omissions or duplications of rows in the results set. For example, if EMPNUM can be null, this DBSLICE= specification omits rows, creating an incorrect result set:

DBSLICE= ("EMPNUM<1000" "EMPNUM>=1000")

Here is a correct form:

DBSLICE= ("EMPNUM<1000" "EMPNUM>=1000" "EMPNUM IS NULL")

In this example, DBSLICE= creates an incorrect set by duplicating SALES with a value of 0.

DBSLICE= ("SALES<0 or SALES=NULL" "SALES>=0")

server

Identifies a particular server node in a DB2 partitioned database or in a Microsoft SQL Server partitioned view. Use this to obtain the best possible Read performance so that your SAS thread can connect directly to the node that contains the data partition that corresponds to your WHERE clause. For DBMS-specific details, see DB2 under UNIX and PC Hosts on page 513 and ODBC on page 743.

Details

If your table reference is eligible for threaded Reads (that is, if it is a read-only LIBNAME table reference), DBSLICE= forces a threaded Read to occur. This partitions the table with the WHERE clauses that you supply. Use DBSLICE= when SAS is unable to generate threaded Reads automatically, or if you can provide better partitioning.

DBSLICE= is appropriate for experienced programmers familiar with the layout of their DBMS tables. A well-tuned DBSLICE= specification usually outperforms SAS automatic partitioning. For example, a well-tuned DBSLICE= specification might better distribute data across threads by taking advantage of a column that SAS/ACCESS cannot use when it automatically generates partitioning WHERE clauses.

DBSLICE= delivers optimal performance for DB2 under UNIX and for Microsoft SQL Server. Conversely, DBSLICE= can degrade performance compared to automatic partitioning. For example, Teradata starts the FastExport Utility for automatic partitioning. If DBSLICE= overrides this action, WHERE clauses are generated instead. Even with well planned WHERE clauses, performance is degraded because FastExport is considerably faster.
Examples

**Example 1: Partition a Column (Two Threads)**
In this example, DBSLICE= partitions on the GENDER column can have only the values m, M, f, and F. This DBSLICE= clause does not work for all DBMSs due to the use of UPPER and single quotation marks. Some DBMSs require double quotation marks around character literals. Two threads are created.

```sas
proc reg SIMPLE
data=lib.customers(DBSLICE="UPPER(GENDER)='M'" "UPPER(GENDER)='F'"));
var age weight;
where years_active>1;
run;
```

**Example 2: Partition a Column (Three Threads)**
The next example partitions on the non-null column CHILDREN, the number of children in a family. Three threads are created.

```sas
data local;
set lib.families(DBSLICE="CHILDREN<2" "CHILDREN>2" "CHILDREN=2"));
where religion="P";
run;
```

---

**DBSLICE= Data Set Option**

Controls the scope of DBMS threaded Reads and the number of DBMS connections.

**Valid in:** DATA and PROC Steps (when accessing DBMS data using SAS/ACCESS software) (also available as a SAS configuration file option, SAS invocation option, global SAS option, and LIBNAME option)

**Defaults:**
- NONE [Vertica]
- THREADED_APPS, none [Greenplum, HAWQ]
- THREADED_APPS,2 [DB2 under z/OS, Oracle, and Teradata]
- THREADED_APPS,2 or THREADED_APPS,3 [DB2 under UNIX and PC Hosts, Informix, Microsoft SQL Server, ODBC, SAP HANA, Sybase, Sybase IQ]

**Data source:**
- DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, ODBC, Oracle, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:**
Support for Vertica was added for SAS 9.4. Support for SAP HANA was added in the first maintenance release for SAS 9.4. Support for Greenplum was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:**
- DBSLICE= data set option, DBSLICE= LIBNAME option, DBSLICE= system option, SLEEP= LIBNAME option, SLEEP= data set option, TENACITY= LIBNAME option, TENACITY= data set option

**Syntax**

```
DBSLICE=NONE | THREADED_APPS | ALL

DBSLICE=( NONE | THREADED_APPS | ALL < max-threads> )
```
DBSLICEPARM=( NONE | THREADED_APPS | ALL<, max-threads> )

Syntax Description

NONE
disables DBMS threaded Reads. SAS reads tables on a single DBMS connection, as it did with SAS 8 and earlier.

THREADED_APPS
makes fully threaded SAS procedures (threaded applications) eligible for threaded Reads.

ALL
makes all read-only librefs eligible for threaded Reads. It includes SAS threaded applications, the SAS DATA step, and numerous SAS procedures.

max-threads
a positive integer value that specifies the maximum number of connections per table read. The second parameter of the option determines the number of threads to read the table in parallel. The number of partitions on the table determine the number of connections made to the Oracle server for retrieving rows from the table. A partition or portion of the data is read on each connection. The combined rows across all partitions are the same regardless of the number of connections. That is, changes to the number of connections do not change the result set. Increasing the number of connections instead redistributes the same result set across more connections.

There are diminishing returns when increasing the number of connections. With each additional connection, more burden is placed on the DBMS, and a smaller percentage of time is saved in SAS. See the DBMS-specific reference section about threaded Reads for your interface before using this parameter.

Details

You can use DBSLICEPARM= in numerous locations. The usual rules of option precedence apply: A table option has the highest precedence, followed by a LIBNAME option, and so on. A SAS configuration file option has the lowest precedence because DBSLICEPARM= in any of the other locations overrides that configuration setting.

DBSLICEPARM=ALL and DBSLICEPARM=THREADED_APPS make SAS programs eligible for threaded Reads. To determine whether threaded Reads are actually generated, turn on SAS tracing and run a program, as shown in this example.

```sas
options sastrace=",,d" sastraceloc=saslog nostsuffix;
proc print data=lib.dbtable(dbsliceparms=(ALL));
   where dbcol>1000;
run;
```

If you want to directly control the threading behavior, use the DBSLICE= data set option.

Greenplum, HAWQ: There is no default value for the maximum number of connections per table read. This value depends on the number of partitions in a table and the arguments that are used with the MOD function in a WHERE clause. For more information, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

DB2 under UNIX and PC Hosts, Informix, Microsoft SQL Server, ODBC, Sybase, Sybase IQ: The default thread number depends on whether an application passes in the number of threads (CPUCOUNT=) and whether the data type of the column that was selected for purposes of data partitioning is binary.
Examples

**Example 1: Disable Threaded Reads for All SAS Users**
Here is how to use DBSLICEPARM= in a SAS configuration file entry in Windows to turn off threaded Reads for all SAS users.

```
-dbsliceparm NONE
```

**Example 2: Enable Threaded Reads for Read-Only References**
Here is how you can use DBSLICEPARM= as a z/OS invocation option to turn on threaded Reads for read-only references to DBMS tables throughout a SAS job.

```
sas o(dbsliceparm=ALL)
```

**Example 3: Increase Maximum Threads (as a SAS Global Option)**
In this example, you can use DBSLICEPARM= as a SAS global option to increase maximum threads to three for SAS threaded applications. Most likely, you would use it as one of the first statements in your SAS code.

```
option dbsliceparm=(threaded_apps,3);
```

**Example 4: Enable Threaded Reads for References Using a Particular Libref**
You can use DBSLICEPARM= as a LIBNAME option to turn on threaded Reads for read-only table references that use this particular libref, as shown in this example

```
libname dblib oracle user=myusr1 password=mypwd1 dbsliceparm=ALL;
```

**Example 5: Enable Threaded Reads as a Table-Level Option**
Here is how to use DBSLICEPARM= as a table-level option to turn on threaded Reads for this particular table, requesting up to four connections.

```
proc reg SIMPLE;
   data=dblib.customers (dbsliceparm=(all,4));
   var age weight;
   where years_active>1;
run;
```

**DBTYPE= Data Set Option**
Specifies a data type to use instead of the default DBMS data type when SAS creates a DBMS table.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** DBMS-specific
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** DBCREATE_TABLE_OPTS= data set option, DBFORCE= data set option, DBNULL= data set option
Syntax

\[
\text{DBTYPE} = (column-name-1='DMBS-type' < ...column-name-n='DMBS-type' > )
\]

Syntax Description

column-name
specifies a DBMS column name.

DBMS-type
specifies a DBMS data type. See the DBMS-specific reference section for your SAS/ACCESS interface for the default data types for your DBMS.

Details

By default, the SAS/ACCESS interface for your DBMS converts each SAS data type to a predetermined DBMS data type when it outputs data to your DBMS. When you need a different data type, use DBTYPE= to override the default data type chosen by the SAS/ACCESS engine.

You can also use this option to specify column modifiers. The allowable syntax for these modifiers is generally DBMS-specific. For more information, see the SQL reference for your database.

MySQL: All text strings are passed as is to the MySQL server. MySQL truncates text strings to fit the maximum length of the field without generating an error message.

Teradata: You can use DBTYPE= to specify data attributes for a column. See your Teradata CREATE TABLE documentation for information about the data type attributes that you can specify. If you specify DBNULL=NO for a column, do not also use DBTYPE= to specify NOT NULL for that column. If you do, NOT NULL is inserted twice in the column definition. This causes Teradata to generate an error message.

Vertica: The default is none.

Examples

Example 1: Specify Data Types for Columns

In this example, DBTYPE= specifies the data types to use when you create columns in the DBMS table.

```
data mydblib.newdept(dbtype=(deptno='number(10,2)' city='char(25)'));
  set mydblib.dept;
run;
```

Example 2: Specify Data Types for Columns in a New Table

This example creates a new Teradata table, NEWDEPT, specifying the Teradata data types for the DEPTNO and CITY columns.

```
data mydblib.newdept(dbtype=(deptno='byteint' city='char(25)'));
set dept;
run;
```

Example 3: Specify a Data Type for a Column in a New Table

This example creates a new Teradata table, NEWEMPLOYEES, and specifies a data type and attributes for the EMPNO column. The example encloses the Teradata type and
attribute information in double quotation marks. Single quotation marks conflict with single quotation marks that the Teradata FORMAT attribute requires. If you use single quotation marks, SAS returns syntax error messages.

data mydblib.newemployees(dbtype= {emno="SMALLINT FORMAT '9(5)'
   CHECK (emno >= 100 AND emno <= 2000)"});
set mydblib.employees;
run;

Example 4: Create a Primary Key for a New Table
Where x indicates the Oracle engine, this example creates a new table, ALLACCTX, and uses DBTYPE= to create the primary key, ALLACCT_PK.

data x.ALLACCTX ( dbtype=(
   SourceSystem = 'varchar(4)'
   acctnum = 'numeric(18,5) CONSTRAINT "ALLACCT_PK" PRIMARY KEY'
   accttype = 'numeric(18,5)'
   balance = 'numeric(18,5)'
   clientid = 'numeric(18,5)'
   closedate = 'date'
   opendate = 'date'
   primary_cd = 'numeric(18,5)'
   status = 'varchar(1)'
) );
set work.ALLACCT ;
format CLOSEDATE date9.;
format OPENDATE date9.;
run;

The code generates this CREATE TABLE statement.

Output 12.1 Output from a CREATE TABLE Statement That Uses DBTYPE= to Specify a Column Modifier

```
CREATE TABLE ALLACCTX(SourceSystem varchar(4),
   acctnum numeric(18,5) CONSTRAINT "ALLACCT_PK" PRIMARY KEY,
   accttype numeric(18,5),balance numeric(18,5),clientid numeric(18,5),
   closedate date,opendate date,primary_cd numeric(18,5),status varchar(1))
```

DEGREE= Data Set Option

Determines whether DB2 uses parallelism.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** ANY

**Data source:** DB2 under z/OS

**See:** DEGREE= LIBNAME option

**Syntax**

```
DEGREE=ANY | 1
```
Syntax Description

**ANY**

enables DB2 to use parallelism, and issues the `SET CURRENT DEGREE = 'xxx'` for all DB2 threads that use that libref.

**1**

explicitly disables the use of parallelism.

Details

When `DEGREE=ANY`, DB2 has the option of using parallelism, when it is appropriate.

Setting `DEGREE=1` prevents DB2 from performing parallel operations. Instead, DB2 is restricted to performing one task that, although this is perhaps slower, it uses fewer system resources.

---

**DIMENSION= Data Set Option**

Specifies whether the database creates dimension tables or fact tables.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Data source:** Aster
- **See:** `DIMENSION= LIBNAME option`, `PARTITION_KEY= LIBNAME option`, `PARTITION_KEY= data set option`

**Syntax**

`DIMENSION= YES | NO`

**Syntax Description**

**YES**

specifies that the database creates dimension tables.

**NO**

specifies that the database creates fact tables.

---

**DISTRIBUTE_ON= Data Set Option**

Specifies a column name to use in the `DISTRIBUTE ON` clause of the CREATE TABLE statement.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Alias:** `DISTRIBUTE= [Netezza]`
- **Default:** none
- **Data source:** Aster, Netezza

**Syntax**

`DISTRIBUTE_ON= 'column-1 <…,column-n>' | RANDOM`
Syntax Description

column-name
    specifies a DBMS column name.

RANDOM
    specifies that data is distributed evenly. For Netezza, the Netezza Performance
    Server does this across all SPUs. This is known as round-robin distribution.

Details

You can use this option to specify a column name to use in the DISTRIBUTE ON=
clause of the CREATE TABLE statement. Each table in the database must have a
distribution key that consists of one to four columns. If you do not specify this option,
the DBMS selects a distribution key.

Examples

Example 1: Create a Distribution Key on a Single Column

proc sql;
create table netlib.customtab(DISTRIBUTE_ON='partno')
    as select partno, customer, orderdat from saslib.orders;
quit;

Example 2: Create a Distribution Key on Many Columns

For more than one column, separate the columns with commas.

data netlib.mytab(DISTRIBUTE_ON='col1,col2');
col1=1;col2=12345;col4='mytest';col5=98.45;
run;

Example 3: Use the RANDOM Keyword

data netlib.foo(distribute_on=random);
mycol1=1;mycol2='test';
run;

DISTRIBUTED_BY= Data Set Option

Uses one or multiple columns to distribute table rows across database segments.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: RANDOMLY DISTRIBUTED
Data source: Greenplum, HAWQ
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

Syntax

DISTRIBUTED_BY=column-1 |…,column-n> | DISTRIBUTED RANDOMLY
**Syntax Description**

*column-name*

specifies a DBMS column name.

**DISTRIBUTED RANDOMLY**

determines the column or set of columns that the Greenplum database uses to distribute table rows across database segments. This is known as round-robin distribution.

**Details**

For uniform distribution—namely, so that table records are stored evenly across segments (machines) that are part of the database configuration—the distribution key should be as unique as possible.

**Example: Create a Table By Specifying a Distribution Key**

```plaintext
libname x greenplm user=myusr1 password=mypwd1 dsn=mysrv1;
data x.sales (dbtype=(id=int qty=int amt=int)
    distributed_by='distributed by (id)');
  id = 1;
  qty = 100;
  sales_date = '27Aug2009'd;
  amt = 20000;
run;
```

It creates the SALES table.

```plaintext
CREATE TABLE SALES
(id int,
 qty int,
 sales_date double precision,
 amt int
) distributed by (id)
```

---

**ERRLIMIT= Data Set Option**

Specifies the number of errors that are allowed before SAS stops processing and issues a rollback.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** 1

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** `DBCOMMIT= LIBNAME option`, `DBCOMMIT= data set option`, `ERRLIMIT= LIBNAME option`, `ML_CHECKPOINT= data set option`

---

**Syntax**

```plaintext
ERRLIMIT=integer
```
**Syntax Description**

integer

specifies a positive integer that represents the number of errors after which SAS stops processing and issues a rollback.

**Details**

SAS ends the step abnormally and calls the DBMS to issue a rollback after a specified number of errors while processing inserts, deletes, updates, and appends. If ERRLIMIT=0, SAS processes all rows no matter how many errors occur. The SAS log displays the total number of rows that SAS processed and the number of failed rows, if applicable.

If the step ends abnormally, any rows that SAS successfully processed after the last commit are rolled back and are therefore lost. Unless DBCOMMIT=1, it is very likely that rows can be lost. The default value is 1000.

*Note:* A significant performance impact can result if you use this option from a SAS client session in SAS/SHARE or SAS/CONNECT environments to create or populate a newly created table. To prevent this, use the default setting, ERRLIMIT=1.

*Teradata:* A rollback to the last checkpoint does not take place on reaching ERRLIMIT because the rows without errors have already been sent to Teradata.

**Example: Specify the Number of Allowable Errors**

In this example, SAS stops processing and issues a rollback to the DBMS at the occurrence of the tenth error. The MYDBLIB libref was assigned in a prior LIBNAME statement.

```sas
data mydblib.employee3 (errlimit=10);
  set mydblib.employees;
  where salary > 40000;
run;
```

---

**ESCAPE_BACKSLASH= Data Set Option**

Specifies whether backslashes in literals are preserved during data copy from a SAS data set to a table.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** NO

**Data source:** MySQL

**See:** ESCAPE_BACKSLASH= LIBNAME option

**Syntax**

ESCAPE_BACKSLASH=YES | NO
Syntax Description

**YES**

specifies that an additional backslash is inserted in every literal value that already contains a backslash.

**NO**

specifies that backslashes that exist in literal values are not preserved. An error results.

Details

MySQL uses the backslash as an escape character. When data that is copied from a SAS data set to a MySQL table contains backslashes in literal values, the MySQL interface can preserve them if `ESCAPE_BACKSLASH=YES`.

Examples

**Example 1: Preserve Backslashes**

In this example, SAS preserves the backslashes for \textit{x} and \textit{y} values.

```sas
libname out mysql user=myusr1 pw=mypwd1
   server=striper database=test port=3306;

data work.test;
   length x y z $10;
   x = "ABC";
   y = "DEF\";
   z = 'GHI\';
run;
data out.test(escape_backslash=yes);
set work.test;
run;
```

The code successfully generates this INSERT statement.

`INSERT INTO 'test' ('x','y','z')  VALUES ('ABC','DEF\\','GHI\\')`

**Example 2: Use the Default Value (Do Not Preserve Backslashes)**

For the prior example, here is the error that is displayed if `ESCAPE_BACKSLASH=NO`.

**ERROR:** Execute error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "GHI\" at line 1

**FASTEXPORT= Data Set Option**

Specifies whether the SAS/ACCESS engine uses the TPT API to read data.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Access
- **Default:** NO
- **Data source:** Teradata, PostgresSQL
FASTEXPORT= Data Set Option

Syntax

FASTEXPORT= YES | NO

Syntax Description

YES

specifies that the FASTEXPORT data set option is to be used.

NO

specifies that the FASTEXPORT data set option is not to be used.

Details

By using the TPT API, you can read data from a Teradata table without working directly with the stand-alone Teradata FastExport utility. When FASTEXPORT=YES, SAS uses the TPT API export driver for bulk reads. If SAS cannot use the TPT API—due to an error or because it is not installed on the system—it still tries to read the data. However, it does not produce an error. To check whether SAS used the TPT API to read data, look for this message in the SAS log:

```
NOTE: Teradata connection: TPT FastExport has read n row(s).
```

When you specify a query band on this option, you must set the DBSLICEPARM= LIBNAME option. The query band is passed as a SESSION query band to the FastExport utility.

To see whether threaded reads are actually generated, turn on SAS tracing by setting OPTIONS SASTRACE=",,d" in your program.

Example

In this example, the TPT API reads SAS data from a Teradata table. SAS still tries to read data even if it cannot use the TPT API.

```
Libname tera Teradata user=myusr1 pw=mypwd1 FASTEXPORT=YES;
/* Create data */
Data tera.testdata;
Do i=1 to 100;
  Output;
End;
Run;
/* Read using FastExport TPT. This note appears in the SAS log if SAS uses TPT.
NOTE: Teradata connection: TPT FastExport has read n row(s).*/
Data work.testdata;
Set tera.testdata;
```
**FASTLOAD= Data Set Option**

Enables you to specify whether this is to be a FASTLOAD or not.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>NO</td>
</tr>
</tbody>
</table>

Data source: Teradata and PostgresSQL

See: BULKLOAD= LIBNAME option, BULKLOAD= data set option, DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option, DBSLICEPARM= system option, FASTLOAD=LIBNAME option LOGDB= LIBNAME option, FASTEXPORT= LIBNAME option Maximizing Teradata Load Performance, MULTILOAD= data set option, QUERY_BAND= LIBNAME option, QUERY_BAND= data set option, SLEEP= LIBNAME option, SLEEP= data set option, TENACITY= LIBNAME option, TENACITY= data set option

**Syntax**

`FASTLOAD YES | NO`

**Syntax Description**

**YES** specifies that the SAS/ACCESS engine use the FastLoad protocol to load the data.

**NO** specifies that the SAS/ACCESS engine does not use the FastLoad protocol to load the data.

**Details**

You can specify FASTLOAD= by using a data set or LIBNAME option. The disadvantage to using the LIBNAME option is that, whenever you insert data from SAS into Teradata tables in the SAS library, the FastLoad protocol is used. This takes Teradata utility slots, which might also cause other load jobs to fail.

Setting the FASTLOAD= data set option on a SAS library hides it from users. In a business intelligence environment, it might be difficult to determine whether this option is set. If you are setting up SAS libraries that only ETL jobs use, it might be acceptable to use the LIBNAME option.

**Note:** A best practice recommendation is to use FASTLOAD= as a data set option unless you have a compelling reason to use it as a LIBNAME option.

```sas
libname mytera TERADATA server=teraserv user=bob pw=bob1;
/* Create and load a table using a Data step. SAS numeric is */
/* forced to be an INTEGER. */
data mytera.table0 (FASTLOAD=YES DBTYPE= (x= INTEGER));
do x = 1 to 1000000;
  output;
end; run;
/* Load an existing table using PROC APPEND. The table must */
```
PROC SQL;
    CONNECT TO TERADATA (USER=bob PW=bob1 SERVER=tera5500);
    EXECUTE (DROP TABLE loadThisTable) by TERADATA;
    EXECUTE (COMMIT) BY TERADATA;
    EXECUTE (CREATE MULTISET TABLE loadThisTable ( a INTEGER , b CHAR(10))
        PRIMARY INDEX (a)) by TERADATA;
    EXECUTE (COMMIT) BY TERADATA;
    QUIT;
DATA work.loadData;
    FORMAT b $10.;
    a = 1;
    output;
    b = 'One';
    output;
    a = 2;
    output;
    b = 'Two';
    output;
    a = 3;
    output;
    b = 'Three';
    output;
RUN;
libname mytera teradata server=teraserv user=bob pw=bob1;
PROC APPEND BASE=mytera.loadThisTable (FASTLOAD=YES BL_LOG=BOB_APPEND_ERR) DATA=work.loadData;
RUN;

FETCH_IDENTITY= Data Set Option

Returns the value of the last inserted identity value.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Data source: DB2 under UNIX and PC Hosts
See: FETCH_IDENTITY= LIBNAME option, INSERTBUFF= data set option

Syntax

FETCH_IDENTITY=YES | NO

Syntax Description

YES
    returns the value of the last inserted identity value.

NO
    disables this option.
Details

You can use this option instead of issuing a separate SELECT statement after an INSERT statement. If FETCH.IDENTITY=YES and the INSERT that is executed is a single-row INSERT, the engine calls the DB/2 identity_val_local() function and places the results into the SYSDB2_LAST.IDENTITY macro variable. Because the DB2 engine default is multirow inserts, you must set INSERTBUFF=1 to force a single-row INSERT.

HDFS_PRINCIPAL= Data Set Option

Specifies the Kerberos principal for HDFS.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS)
- **Default:** none
- **Restriction:** This option applies only when you configure HDFS to allow Kerberos authentication.
- **Requirement:** To specify this option, you must first set BULKLOAD=YES.
- **Data source:** Impala
- **Note:** Support for this data set option was added in the second maintenance release for SAS 9.4.
- **See:** BULKLOAD= LIBNAME option, IMPALA_PRINCIPAL= LIBNAME option

**Syntax**

```
HDFS_PRINCIPAL='principal'
```

**Required Argument**

*principal*

specifies the server’s Kerberos service principal name (SPN). Surround the principal value with single or double quotation marks. For example, you might specify a principal value that is similar to the following code:

```
hdfs_principal='hdfs/hdfs_host.example.com@TEST.EXAMPLE.COM'
```

**IGNORE_READ_ONLY_COLUMNS= Data Set Option**

Specifies whether to ignore or include columns whose data types are read-only when generating an SQL statement for inserts or updates.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, PostgreSQL, SAP HANA, Sybase IQ, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** IGNORE_READ_ONLY_COLUMNS=
Syntax

**IGNORE_READ_ONLY_COLUMNS**=YES | NO

**Syntax Description**

**YES**

specifies that the SAS/ACCESS engine ignores columns whose data types are read-only when you are generating insert and update SQL statements

**NO**

specifies that the SAS/ACCESS engine does not ignore columns whose data types are read-only when you are generating insert and update SQL statements

**Details**

Several databases include data types that can be read-only, such as the Microsoft SQL Server timestamp data type. Several databases also have properties that allow certain data types to be read-only, such as the Microsoft SQL Server identity property.

When **IGNORE_READ_ONLY_COLUMNS**=NO (the default) and a DBMS table contains a column that is read-only, an error is returned that the data could not be modified for that column.

**Example: Insert Data into a Table**

For this example, a database that contains the table Products is created with two columns: ID and PRODUCT_NAME. The ID column is defined by a read-only data type and PRODUCT_NAME is a character column.

```
CREATE TABLE products (id int IDENTITY PRIMARY KEY, product_name varchar(40))
```

If you have a SAS data set that contains the name of your products, you can insert the data from the SAS data set into the Products table.

```
data work.products;
  id=1;
  product_name='screwdriver';
  output;
  id=2;
  product_name='hammer';
  output;
  id=3;
  product_name='saw';
  output;
  id=4;
  product_name='shovel';
  output;
run;
```

When **IGNORE_READ_ONLY_COLUMNS**=NO (the default), the database returns an error because the ID column cannot be updated. However, if you set the option to **YES** and execute a PROC APPEND, the append succeeds and the generated SQL statement does not contain the ID column.

```
libname x odbc uid=myusr1 pwd=mypwd1 dsn=lupinss
   ignore_read_only_columns=yes;
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc append base=x.PRODUCTS data=work.products;
```
IN= Data Set Option

Lets you specify the database or table space in which you want to create a new table.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Alias: TABLESPACE=
Default: LIBNAME option setting
Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS
See: IN= LIBNAME option

Syntax

IN=’database-name.tablespace-name’ | ’DATABASE database-name’

Syntax Description

database-name.tablespace-name
specifies the names of the database and table space, which are separated by a period.

DATABASE database-name
specifies only the database name. In this case, you specify the word DATABASE, a space, and the database name. Enclose the entire specification in single quotation marks.

Details

The IN= option is relevant only when you are creating a new table. If you omit this option, the default is to create the table in the default database or table space.

INSERT_SQL= Data Set Option

Determines the method to use to insert rows into a data source.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: LIBNAME option setting
Data source: Microsoft SQL Server, ODBC, OLE DB
See: INSERT_SQL= LIBNAME option, INSERTBUFF= data set option

Syntax

INSERT_SQL= YES | NO

Syntax Description

YES
specifies that the SAS/ACCESS engine uses the data source's SQL insert method to insert new rows into a table.
NO
specifies that the SAS/ACCESS engine uses an alternate (DBMS-specific) method to add new rows to a table.

Details
Flat-file databases such as dBase, FoxPro, and text files have generally improved insert performance when INSERT_SQL=NO. Other databases might have inferior insert performance or might fail with this setting. Therefore, you should experiment to determine the optimal setting for your situation.

Microsoft SQL Server:
The Microsoft SQL Server default is YES. When INSERT_SQL=NO, the SQLSetPos (SQL_ADD) function inserts rows in groups that are the size of the INSERTBUFF= value. The SQLSetPos (SQL_ADD) function does not work unless your ODBC driver supports it.

ODBC:
The default for ODBC is YES, except for Microsoft Access, which has a default of NO. When INSERT_SQL=NO, the SQLSetPos (SQL_ADD) function inserts rows in groups that are the size of the INSERTBUFF= option value. The SQLSetPos (SQL_ADD) function does not work unless your ODBC driver supports it.

OLE DB:
By default, the OLE DB interface attempts to use the most efficient row insertion method for each data source. You can use the INSERT_SQL option to override the default in the event that it is not optimal for your situation. The OLE DB alternate method (used when this option is set to NO) uses the OLE DB IRowsetChange interface.

INSERTBUFF= Data Set Option
Specifies the number of rows in a single DBMS insert.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: LIBNAME option setting
Restrictions: SAS application messages that indicate the success or failure of an Insert operation represent information for only a single insert, even when multiple inserts are performed. Therefore, when you assign a value that is greater than INSERTBUFF=1, these messages might be incorrect.
When you insert rows with the VIEWTABLE window or the FSVIEW or FSEDIT procedure, use INSERTBUFF=1 to prevent the DBMS interface from trying to insert multiple rows. These features do not support inserting more than one row at a time.
Additional driver-specific restrictions might apply.
Interactions: SAS allows the maximum number of rows that the DBMS allows.
If you set DBCOMMIT= with a value that is less than the value of INSERTBUFF=, then DBCOMMIT= overrides INSERTBUFF=.
The optimal value for this option varies with factors such as network type and available memory.
To use this option, you must set INSERT_SQL=YES. (DB2 under UNIX and PC Hosts, Greenplum, Microsoft SQL Server)
If one row in the insert buffer fails, all rows in the insert buffer fail. (DB2 under UNIX and PC Hosts)
Values greater than 0 activate the INSERTBUFF= option, and the engine calculates how many rows it can insert at one time, based on row size. If one row in the insert buffer fails, all rows in the insert buffer might fail, depending on your storage type.

(MySQL)

**Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**Tip:** You might need to experiment with different values to determine the best value for your site.

**See:**
- DBCOMMIT= LIBNAME option
- DBCOMMIT= data set option
- INSERTBUFF= LIBNAME option
- INSERT_BUFF= LIBNAME option
- INSERT_SQL LIBNAME option
- INSERT_SQL data set option
- READBUFF= LIBNAME option
- READBUFF= data set option

**Syntax**

```
INSERTBUFF=positive-integer
```

**Syntax Description**

- `positive-integer` specifies the number of rows to insert. SAS allows the maximum that the DBMS allows.

---

**KEYSET_SIZE= Data Set Option**

Specifies the number of rows in the cursor that the keyset drives.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** LIBNAME option setting
- **Requirement:** This option is valid only when CURSOR_TYPE=KEYSET_DRIVEN.
- **Data source:** Microsoft SQL Server, ODBC
- **See:** KEYSET_SIZE= LIBNAME option

**Syntax**

```
KEYSET_SIZE=number-of-rows
```

**Syntax Description**

- `number-of-rows` specifies a positive integer from 0 through the number of rows in the cursor.

**Details**

If `KEYSET_SIZE`=0, the entire cursor is keyset-driven.

If you specify a value greater than 0, that value indicates the number of rows within the cursor that function as a keyset-driven cursor. When you scroll beyond the bounds that
KEYSET_SIZE= specifies, the cursor becomes dynamic and new rows might be included in the cursor. This results in a new keyset, where the cursor functions as a keyset-driven cursor again.

When you specify a value between 1 and the number of rows in the cursor, the cursor is considered to be a mixed cursor. Part of the cursor functions as a keyset-driven cursor, and another part of the cursor functions as a dynamic cursor.

### LOCATION= Data Set Option

Lets you further specify exactly where a table resides.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Alias:** LOC=
- **Default:** LIBNAME option setting
- **Requirement:** If you specify LOCATION=, you must also specify the AUTHID= data set option.
- **Data source:** DB2 under z/OS
- **See:** AUTHID= data set option, LOCATION= LIBNAME option

#### Syntax

LOCATION=location-name

#### Details

The location name maps to the location in the SYSIBM.LOCATIONS catalog in the communication database.

In SAS/ACCESS Interface to DB2 under z/OS, the location is converted to the first level of a three-level table name: location-name.AUTHID.TABLE. The DB2 Distributed Data Facility (DDF) makes the connection implicitly to the remote DB2 subsystem when DB2 receives a three-level name in an SQL statement.

### LOCKTABLE= Data Set Option

Places exclusive or shared locks on tables.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** LIBNAME option setting
- **Restriction:** If you omit LOCKTABLE=, no locking occurs.
- **Requirement:** You can lock tables only if you are the owner or have been granted the necessary privilege.
- **Data source:** Informix
- **See:** LOCKTABLE= LIBNAME option

#### Syntax

LOCKTABLE=EXCLUSIVE | SHARE
**Syntax Description**

**EXCLUSIVE**
locks a table exclusively, preventing other users from accessing any table that you open in the libref.

**SHARE**
locks a table in shared mode. It allows other users or processes to read data from the tables but preventing users from updating data.

---

**MBUFSIZE= Data Set Option**

Specifies the size of the shared memory buffers to use for transferring data from SAS to Teradata.

**Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

**Default:** 64K

**Requirement:** To specify this option, you must first set MULTILOAD=YES.

**Data source:** Teradata

**Tip:** Use BUFFERS= to vary the number of buffers for data transfer from 1 to 8. Use MBUFSIZE= to vary the size of the shared memory buffers from the size of each data row up to 1MB.

**See:** BUFFERS= data set option, MULTILOAD= data set option, Teradata MultiLoad documentation, Using MultiLoad

---

**Syntax**

```
MBUFSIZE=size-of-shared-memory-buffers
```

**Syntax Description**

`size-of-shared-memory-buffers`
specifies a numeric value (between the size of a row being loaded and 1MB) that specifies the buffer size.

**Details**

Two data set options are available for tuning the number and size of data buffers used for transferring data from SAS to Teradata.

When you use MULTILOAD=, data transfers from SAS to Teradata using shared memory segments. The default shared memory buffer size is 64K. The default number of shared memory buffers that are used for the transfer is 2.

---

**ML_CHECKPOINT= Data Set Option**

Specifies the interval between checkpoint operations in minutes.

**Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

**Default:** none (see “Details”)
**Requirement:**
To specify this option, you must first set MULTILOAD=YES.

**Data source:**
Teradata

**See:**
DBCOMMIT= LIBNAME option, DBCOMMIT= data set option, MULTILOAD= data set option, Teradata MultiLoad documentation, Using MultiLoad

### Syntax

ML_CHECKPOINT=checkpoint-rate

**Syntax Description**

**checkpoint-rate**

a numeric value that specifies the interval between checkpoint operations in minutes.

### Details

If you do not specify a value for ML_CHECKPOINT=, the Teradata Multiload default of 15 applies. If ML_CHECKPOINT= is between 1 and 59 inclusive, checkpoints are taken at the specified intervals, in minutes. If ML_CHECKPOINT= is greater than or equal to 60, a checkpoint occurs after a multiple of the specified rows is loaded.

ML_CHECKPOINT= functions very similarly to CHECKPOINT in the native Teradata MultiLoad utility. However, it differs from DBCOMMIT=. Both DBCOMMIT= and its alias, CHECKPOINT, are disabled to prevent any conflict.

---

**ML_ERROR1= Data Set Option**

Specifies the name of a temporary table that MultiLoad uses to track errors that were generated during the acquisition phase of bulk loading.

**Valid in:**
DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

**Default:**
none

**Restriction:**
Do not use ML_ERROR1 with the ML_LOG= data set option. ML_LOG= provides a common prefix for all temporary tables that the Teradata MultiLoad utility uses.

**Requirements:**
To specify this option, you must first set MULTILOAD=YES.

When you restart a failed MultiLoad job, you must specify the same acquisition table from the earlier run so that the MultiLoad job can restart correctly. Using ML_RESTART=, ML_ERROR2=, and ML_WORK=, you must also specify the same log table, application error table, and work table upon restarting.

**Data source:**
Teradata

**See:**
ML_ERROR2= data set option, ML_LOG= data set option, ML_RESTART= data set option, ML_WORK= data set option, MULTILOAD= data set option, Teradata MultiLoad documentation, Using MultiLoad

### Syntax

ML_ERROR1= temporary-table-name
**Syntax Description**

temporary-table-name

specifies the name of a temporary table that MultiLoad uses to track errors that were generated during the acquisition phase of bulk loading.

**Details**

Use this option to specify the name of a table to use for tracking errors that were generated during the acquisition phase of MultiLoad bulk loading. By default, the acquisition error table is named SAS_ML_ET_randnum, where randnum is a random number.

---

**ML_ERROR2= Data Set Option**

Specifies the name of a temporary table that MultiLoad uses to track errors that were generated during the application phase of bulk loading.

**Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

**Default:** none

**Restriction:** Do not use ML_ERROR2 with ML_LOG=, which provides a common prefix for all temporary tables that the Teradata MultiLoad utility uses.

**Requirements:** To specify this option, you must first set MULTILOAD=YES. When you restart a failed MultiLoad job, you must specify the same application table from the earlier run so that the MultiLoad job can restart correctly. Using ML_RESTART=, ML_ERROR1=, and ML_WORK=, you must also specify the same log table, acquisition error table, and work table upon restarting.

**Data source:** Teradata

**See:** ML_ERROR1= data set option, ML_LOG= data set option, ML_RESTART= data set option, ML_WORK= data set option, MULTILOAD= data set option, Teradata MultiLoad documentation, Using MultiLoad

---

**Syntax**

ML_ERROR2=**temporary-table-name**

**Syntax Description**

temporary-table-name

specifies the name of a temporary table that MultiLoad uses to track errors that were generated during the application phase of bulk loading.

**Details**

By default, the application error table is named SAS_ML_UT_randnum, where randnum is a random number.
ML_LOG= Data Set Option

Specifies a prefix for the names of the temporary tables that MultiLoad uses during a bulk-loading operation.

**Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

**Default:** none

**Restriction:** Do not use ML_LOG= with ML_RESTART=, ML_ERROR1=, ML_ERROR2=, or ML_WORK= because ML_LOG= provide specific names to the temporary files.

**Requirement:** To specify this option, you must first set MULTILOAD=YES.

**Data source:** Teradata

**See:** ML_ERROR1= data set option, ML_ERROR2= data set option, ML_RESTART= data set option, ML_WORK= data set option, MULTILOAD= data set option, Teradata MultiLoad documentation, Using MultiLoad

**Syntax**

ML_LOG=prefix-for-MultiLoad-temporary-tables

**Syntax Description**

prefix-for-MultiLoad-temporary-tables

specifies the prefix to use when naming Teradata tables that the Teradata MultiLoad utility uses during a bulk-loading operation.

**Details**

The MultiLoad utility uses a log table, two error tables, and a work table while loading data to the target table. By default, here are the names for these tables, where randnum is a random number.

<table>
<thead>
<tr>
<th>Temporary Table</th>
<th>Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart table</td>
<td>SAS_ML_RS_{randnum}</td>
</tr>
<tr>
<td>Acquisition error table</td>
<td>SAS_ML_ET_{randnum}</td>
</tr>
<tr>
<td>Application error table</td>
<td>SAS_ML_UT_{randnum}</td>
</tr>
<tr>
<td>Work table</td>
<td>SAS_ML_WT_{randnum}</td>
</tr>
</tbody>
</table>

To override the default names, here are the table names that would be generated if ML_LOG=MY_LOAD, for example.

<table>
<thead>
<tr>
<th>Temporary Table</th>
<th>Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart table</td>
<td>MY_LOAD_RS</td>
</tr>
</tbody>
</table>
Temporary Table | Table Name
--- | ---
Acquisition error table | MY_LOAD_ET
Application error table | MY_LOAD_UT
Work table | MY_LOAD_WT

SAS/ACCESS automatically deletes the error tables if no errors are logged. If there are errors, the tables are retained, and SAS/ACCESS issues a warning message that includes the names of the tables in error.

**ML_RESTART= Data Set Option**

Specifies the name of a temporary table that MultiLoad uses to track checkpoint information.

- **Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)
- **Default:** none
- **Restriction:** Do not use ML_RESTART= with ML_LOG=, which provides a common prefix for all temporary tables that the Teradata MultiLoad utility uses.
- **Requirements:** To specify this option, you must first set MULTILOAD=YES. When you restart a failed MultiLoad job, you must specify the same application table from the earlier run so that the MultiLoad job can restart correctly. Using ML_RESTART=, ML_ERROR1=, and ML_WORK=, you must also specify the same log table, acquisition error table, and work table upon restarting.
- **Data source:** Teradata
- **See:** ML_ERROR1= data set option, ML_ERROR2 data set option, ML_LOG= data set option, ML_WORK= data set option, MULTILOAD= data set option, Teradata MultiLoad documentation, Using MultiLoad

**Syntax**

ML_RESTART= *temporary-table-name*

**Syntax Description**

*temporary-table-name*

  specifies the name of the temporary table that the Teradata MultiLoad utility uses to track checkpoint information.

**Details**

Use this option to specify the name of a table to store checkpoint information. Upon restart, ML_RESTART= is used to specify the name of the log table that you used for tracking checkpoint information in the earlier failed run.
ML_WORK= Data Set Option

Specifies the name of a temporary table that MultiLoad uses to store intermediate data.

Valid in: DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

Default: none

Restriction: Do not use ML_WORK= with ML_LOG=, which provides a common prefix for all temporary tables that the Teradata MultiLoad utility uses.

Requirement: To specify this option, you must first set MULTILOAD=YES.

Data source: Teradata

See: ML_ERROR1= data set option, ML_ERROR2 data set option, ML_LOG= data set option, ML_RESTART= data set option, MULTILOAD= data set option, Teradata MultiLoad documentation, Using MultiLoad

Syntax

ML_WORK=temporary-table-name

Syntax Description

temporary-table-name

specifies the name of a temporary table that MultiLoad uses to store intermediate data that the MultiLoad utility receives during bulk loading.

Details

Use this option to specify the name of the table to use for tracking intermediate data that the MultiLoad utility received during bulk loading. When you restart the job, use ML_WORK= to specify the name of the table for tracking intermediate data during a previously failed MultiLoad job.

MULTILOAD= Data Set Option

Specifies whether Teradata Insert and Append operations should use the Teradata MultiLoad utility.

Valid in: DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

Default: NO

Restriction: You cannot use ML_LOG with any of these options: ML_ERROR1, ML_ERROR2, ML_WORK, and ML_RESTART.

Data source: Teradata

See: BUFFERS= data set option, BULKLOAD= LIBNAME option, BULKLOAD= data set option, DBCOMMIT= LIBNAME option, DBCOMMIT= data set option, FASTEXPORT= LIBNAME option, Maximizing Teradata Load Performance, MBUFSIZE= data set option, ML_CHECKPOINT= data set option, ML_ERROR1= data set option, ML_ERROR2= data set option, ML_LOG= data set option, ML_RESTART= data set option, ML_WORK= data set option, QUERY_BAND=
LIBNAME option, QUERY_BAND= data set option, SLEEP= data set option, TENACITY= data set option, UPSERT= data set option, UPSERT_WHERE= data set option, UPSERT_CONDITION= data set option, Teradata MultiLoad documentation, Using MultiLoad

Syntax

MULTILOAD=YES | NO

Syntax Description

YES
uses the Teradata MultiLoad utility, if available, to load Teradata tables.

NO
 sends inserts to Teradata tables one row at a time.

Details

Bulk Loading

The SAS/ACCESS MultiLoad facility provides a bulk-loading method of loading both empty and existing Teradata tables. Unlike FastLoad, MultiLoad can append data to existing tables.

To determine whether threaded Reads are actually generated, turn on SAS tracing by setting OPTIONS SASTRACE=",d" in your program.

Data Buffers

Two data set options are available for tuning the number and the size of data buffers that are used for transferring data from SAS to Teradata. Data is transferred from SAS to Teradata using shared memory. The default shared memory buffer size is 64K. The default number of shared memory buffers used for the transfer is 2.

You can use BUFFERS= to vary the number of buffers for data transfer from 1 to 8. You can use MBUFSIZE= to vary the size of the shared memory buffers from the size of each data row up to 1MB.

Temporary Tables

The Teradata MultiLoad utility uses four different temporary tables when it performs the bulk-loading operation. It uses a log table to track restart information, two error tables to track errors, and a work table to hold data before the Insert operation is made.

By default, the SAS/ACCESS MultiLoad facility generates names for these temporary tables, where randnum represents a random number. To specify a different name for these tables, use ML_RESTART=, ML_ERROR1=, ML_ERROR2=, and ML_WORK=, respectively.

<table>
<thead>
<tr>
<th>Temporary Table</th>
<th>Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restart table</td>
<td>SAS_ML_RS_randnum</td>
</tr>
<tr>
<td>Acquisition error table</td>
<td>SAS_ML_ET_randnum</td>
</tr>
<tr>
<td>Application error table</td>
<td>SAS_ML_UT_randnum</td>
</tr>
</tbody>
</table>
### Temporary Table

<table>
<thead>
<tr>
<th>Work table</th>
<th>SAS_ML_WT_randnum</th>
</tr>
</thead>
</table>

You can use `ML_LOG=` to specify a prefix for the temporary table names that MultiLoad uses.

Here is the order that is used for naming the temporary tables that MultiLoad uses:

1. If you set `ML_LOG=`, the prefix that you specified is used when naming temporary tables for MultiLoad.
2. If you do not specify `ML_LOG=`, the values that you specified for `ML_ERROR1`, `ML_ERROR2`, `ML_WORK`, `ML_RESTART` are used.
3. If you do not specify any table naming options, temporary table names are generated by default.

### Restarting MultiLoad

The MultiLoad bulk-loading operation (or MultiLoad job) works in phases. The first is the **acquisition phase**, during which data is transferred from SAS to Teradata work tables. The second is the **application phase**, during which data is applied to the target table.

If the MultiLoad job fails during the acquisition phase, you can restart the job from the last successful checkpoint. The exact observation from which the MultiLoad job must be restarted is displayed in the SAS log. If the MultiLoad job fails in the application phase — when data is loaded onto the target tables from the work table — restart the MultiLoad job outside of SAS. The MultiLoad restart script is displayed in the SAS log. You can run the generated MultiLoad script outside of SAS to complete the load.

You can use `ML_CHECKPOINT=` to specify the checkpoint rate. Specify a value for `ML_CHECKPOINT=` if you want restart capability. If checkpoint tracking is not used and the MultiLoad fails in the acquisition phase, the load needs to be restarted from the beginning. `ML_CHECKPOINT=0` is the default, and no checkpoints are recoded if you use the default.

If `ML_CHECKPOINT` is between 1 and 59 inclusive, checkpoints are recorded at the specified interval in minutes. If `ML_CHECKPOINT` is greater than or equal to 60, a checkpoint occurs after a multiple of the specified rows are loaded.

`ML_CHECKPOINT=` functions very much like the Teradata MultiLoad utility checkpoint, but it differs from `DBCCommit=`.

### Restrictions When Restarting a Failed MultiLoad Job

These restrictions apply when you restart a failed MultiLoad job.

- **Checkpoint rate**: The failed MultiLoad job must have specified a checkpoint rate other than 0 using the `ML_CHECKPOINT=` data set option. Otherwise, restarting begins from the first record of the source data.

Checkpoints are relevant only to the acquisition phase of MultiLoad. Even if `ML_CHECKPOINT=0` is specified, a checkpoint takes place at the end of the acquisition phase. If the job fails in the application phase, restart the job outside of SAS using the MultiLoad script written to the SAS log.

For example, this MultiLoad job takes a checkpoint every 1000 records.

```sas
libname trlib teradata user=myusr1 pw=mypwd1 server=dbc;
/* Create data to MultiLoad */
```
data work.testdata;
  do x=1 to 50000;
    output;
  end;
end;

**Append process:** You must restart the failed MultiLoad job as an append process because the target table already exists. It is also necessary to identify the work tables, restart table, and the error tables used in the original job.

For example, assume that the DATA step shown above failed with this error message in the SAS log:

```
ERROR: MultiLoad failed with DBS error 2644 after a checkpoint was taken for 13000 records. Correct error and restart as an append process with data set options ML_RESTART=SAS_ML_RS_1436199780, ML_ERROR1=SAS_ML_ET_1436199780, ML_ERROR2=SAS_ML_UT_1436199780, and ML_WORK=SAS_ML_WT_1436199780. If the first run used FIRSTOBS=n, use the value (7278+n-1) for FIRSTOBS in the restart. Otherwise, use FIRSTOBS=7278. Sometimes the FIRSTOBS value that is used on the restart can be an earlier position than the last checkpoint because restart is block-oriented and not record-oriented.
```

After you fix the error, you must restart the job as an append process. You must also specify the same work, error, and restart tables as you used in the earlier run. You use a FIRSTOBS= value on the source table to specify the record from which to restart.

```
/* Restart a MultiLoad job that failed in the acquisition phase after correcting the error */
proc append data=work.testdata(FIRSTOBS=7278)
  base=trlib.mlfloat(MultiLoad=YES ML_RESTART=SAS_ML_RS_1436199780
                    ML_ERROR1=SAS_ML_ET_1436199780 ML_ERROR2=SAS_ML_UT_1436199780
                    ML_WORK=SAS_ML_WT_1436199780 ML_CHECKPOINT=1000);
run;
```

**Using ML_LOG=:** If you used ML_LOG= in the run that failed, you can specify the same value for ML_LOG= on restart. Therefore, you do not need to specify four data set options to identify the temporary tables that MultiLoad uses.

For example, assume that this is how the original run used ML_LOG=:

```
data trlib.mlfloat(MultiLoad=yes ML_CHECKPOINT=1000 ML_LOG=MY_ERRORS);
set work.testdata;
run;
```

If this DATA step fails with this error, the restart capability needs only ML_LOG= to identify all necessary tables.

```
ERROR: MultiLoad failed with DBS error 2644 after a checkpoint was taken for 13000 records. Correct error and restart as an append process with data set options ML_RESTART=SAS_ML_RS_1436199780, ML_ERROR1=SAS_ML_ET_1436199780, ML_ERROR2=SAS_ML_UT_1436199780, and ML_WORK=SAS_ML_WT_1436199780. If the first run used FIRSTOBS=n, use the value (7278+n-1) for FIRSTOBS in the restart. Otherwise use FIRSTOBS=7278. Sometimes the FIRSTOBS value used on the restart can be an earlier position than the last checkpoint because restart is block-oriented and not record-oriented.
```
proc append data=work.testdata(FIRSTOBS=7278)
   base=trlib.mlfloat (MultiLoad=yes ML_LOG=MY_ERRORS ML_CHECKPOINT=1000);
run;

• Application-phase failure: If the MultiLoad process fails in the application phase, SAS has already transferred all data to be loaded to Teradata. You must restart a MultiLoad job outside of SAS using the script that is written to the SAS log. See your Teradata documentation on the MultiLoad utility for instructions on how to run MultiLoad scripts. Here is an example of a script that is written in the SAS log.

=-=-=-=-=-=-=-=- MultiLoad restart script starts here ==-=-=-=-=-
.LOGTABLE MY_ERRORS_RS;
.LOGON boom/mloaduser,********;
.begin import mload tables "mlfloat" CHECKPOINT 0 WORKTABLES
   MY_ERRORS_WT ERRORTABLES
      MY_ERRORS_BT MY_ERRORS_UT
/*TIFY HIGH EXIT SASMLNE.DLL TEXT '2180*/;
.layout saslayout indicators;
.PIELD "x" * FLOAT;
.DML Label SASDML;
insert into "mlfloat".*;
.IMPORT INFILE DUMMY
/*SMOD SASMLAM.DLL '2180 2180 2180 */
FORMAT UNFORMAT LAYOUT SASLAYOUT
APPLY SASDML;
.END MLOAD;
.LOGOFF;
=-=-=-=-=-=-=-=- MultiLoad restart script ends here ==-=-=-=-=-
ERROR: MultiLoad failed with DBS error 2644 in the application phase. Run the MultiLoad restart script listed above outside of SAS to restart the job.

• Using a value for FIRSTOBS=: If the original run used a value for FIRSTOBS= for the source data, use the formula from the SAS log error message to calculate the value for FIRSTOBS= upon restart. These examples show how to do this.

/* Create data to MultiLoad */
data work.testdata;
   do x=1 to 50000;
      output;
   end;
run;
libname trlib teradata user=myusr1 pw=mypwd1 server=boom;
/* Load 40,000 rows to the Teradata table */
data trlib.mlfloat (MultiLoad=yes ML_CHECKPOINT=1000 ML_LOG=MY_ERRORS);
set work.testdata(FIRSTOBS=10001);
run;

Assume that the DATA step shown above failed with this error message:

ERROR: MultiLoad failed with DBS error 2644 after a checkpoint was taken for 13000 records. Correct the error and restart the load as an append process with data set option ML_LOG=MY_ERRORS. If the first run used FIRSTOBS=n, use the value (7278+n-1) for FIRSTOBS in the restart. Otherwise use FIRSTOBS=7278. Sometimes the FIRSTOBS value that is specified on the restart can be an earlier position than the last checkpoint because MultiLoad restart is block-oriented and not record-oriented.
The FIRSTOBS for the restart step can be calculated using the formula provided—that is, \( \text{FIRSTOBS} = 7278 + 100001 - 1 = 17278 \). Use FIRSTOBS=17278 on the source data.

```sas
proc append data=work.testdata(FIRSTOBS=17278)
   base=trl.lib.mfloat (MultiLoad=YES
   ML_LOG=MY_ERRORS ML_CHECKPOINT=1000);
run;
```

Please keep these considerations in mind.

- **DBCOMMIT=** is disabled for MultiLoad in order to prevent any conflict with ML_CHECKPOINT=.
- A rollback to the last checkpoint does not take place upon reaching ERRLIMIT= because rows without errors have already been sent to Teradata.
- **How to correctly restart:** For restart to work correctly, the data source must return data in the same order. If the order of data that is read varies between runs and the load job fails in the application phase, delete temporary tables and restart the load as a new process. If the job fails in the application phase, restart the job outside of SAS as usual. You need to do this because the data that is needed to complete the load has already been transferred.
- **Block-oriented versus record-oriented:** The restart capability in MultiLoad is block-oriented, not record-oriented. For example, if a checkpoint was taken at 5000 records, you might need to restart from an earlier record, such as record 4000. You do this because the block of data containing record 5001 might have started at record 4000. The exact record where restart should occur is displayed in the SAS log.

### Examples

**Example 1: Load SAS Data to an Alternate Database**

This example uses MultiLoad to load SAS data to an alternate database. It specifies database=mloaduser in the LIBNAME statement.

```sas
libname trlib teradata user=myusr1 pw=mypwd1
   server=dbc database=mloaduser;
/*MultiLoad 20000 observations into alternate database mloaduser */
data trlib.trmload14 (DBCREATE_TABLE_OPTS="PRIMARY INDEX(IDNUM)"
   MultiLoad=yes ML_LOG=TRMLOAD14 ML_CHECKPOINT=5000);
set permdata.BIG1MIL(drop=year obs=20000);
run;
```

**Example 2: Extract Data from One Table to Another**

This example extracts data from one table using FastExport and loads data into another table using MultiLoad.

```sas
libname trlib teradata user=myusr1 pw=mypwd1 server=dbc;
/* Create data to load */
data trlib.trodd (DBCREATE_TABLE_OPTS="PRIMARY INDEX(IDNUM)"
   MultiLoad=yes);
set permdata.BIG1MIL(drop=year obs=10000);
where mod(IDNUM,2)=1;
run;
/* FastExport from one table and MultiLoad into another */
proc append data=trlib.treven(dbsliceparm=all)
   base=trl.lib.mfloat (MultiLoad=YES
   ML_LOG=MY_ERRORS ML_CHECKPOINT=1000);
run;
```
MULTISTMT= Data Set Option

Specifies whether insert statements are sent to Teradata one at a time or in a group.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Restriction: You cannot currently use MULTISTMT= with the ERRLIMIT= option.
Data source: Teradata
See: DBCOMMIT= LIBNAME option, DBCOMMIT= data set option, ERRLIMIT= LIBNAME option, ERRLIMIT= data set option, MULTILOAD= data set option MULTISTMT= LIBNAME option

Syntax

MULTISTMT=YES | NO

Syntax Description

YES
- tries to send as many inserts to Teradata that can fit in a 64K buffer. If multistatement inserts are not possible, processing reverts to single-row inserts.

NO
- send inserts to Teradata one row at a time.

Details

When you request multi-statement inserts, SAS first determines how many insert statements it can send to Teradata. Several factors determine the actual number of statements that SAS can send—for example:

- how many SQL insert statements can fit in a 64K buffer
- how many data rows can fit in the 64K data buffer
- how many inserts the Teradata server chooses to accept

When you need to insert large volumes of data, you can significantly improve performance by using MULTISTMT= instead of inserting only single-row.

When you also specify DBCOMMIT= , SAS determines the number of insert statements to send together at one time. It uses the smaller of the DBCOMMIT= value and the number of insert statements that can fit in a buffer.

Examples

Example 1: Send and Insert Statements One at a Time
This example shows how to send insert statements one at a time to Teradata.

libname user teradata user=myusr1 pw=XXXXXX server=dbc;
proc datasets library=user;
Example 2: Send 100 Rows at a Time
In this example, DBCOMMIT=100. Therefore, SAS issues a commit after every 100 rows, sending only 100 rows at a time.

```
libname user teradata user=myusr1 pw=XXXXX server=dbc;
proc datasets library=user;
  delete testdata;run;
proc datasets library=user;
  delete testdata;run;
data user.testdata(MULTISTMT=YES DBCOMMIT=100);
do i=1 to 1000;
  output;
end;
run;
```

Example 3: Send a Specified Group of Rows at a Time
In this example, DBCOMMIT=1000, which is much higher than in the previous example. SAS sends as many rows as it can fit in the buffer at a time (up to 1000), and it issues a commit after every 1000 rows. If only 600 can fit, 600 are sent to the database. It is followed by the remaining 400—the difference between 1000 and the initial 600 that were already sent. SAS then commits all rows.

```
libname user teradata user=myusr1 pw=XXXXX server=dbc;
proc datasets library=user;
  delete testdata;run;
proc datasets library=user;
  delete testdata;run;
data user.testdata(MULTISTMT=YES DBCOMMIT=1000);
do i=1 to 10000;
  output;
end;
run;
```

Example 4: Use a Global Options to Store a Temporary Table
This example sets CONNECTION=GLOBAL for all tables, creates a global temporary table, and stores the table in the current database schema.

```
libname user teradata user=myusr1 pw=XXXXX server=dbc connection=global;
proc datasets library=user;
  delete temp1;run;

proc sql;
  connect to teradata(user=myusr1 pw=XXXXXXX server=dbc connection=global);
  execute (CREATE GLOBAL TEMPORARY TABLE temp1 (col1 INT )
           ON COMMIT PRESERVE ROWS) by teradata;
  execute (COMMIT WORK) by teradata;
quilt;
data work.test;
do col1=1 to 1000;
```
output;
end;
run;
proc append data=work.test base=user.temp1(multistmt=yes);
run;

**NULLCHAR= Data Set Option**

Indicates how missing SAS character values are handled during insert, update, DBINDEX=, and DBKEY= processing.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** SAS

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**See:** BULKLOAD= data set option, DBINDEX= data set option, DBKEY= data set option, DBNULL= data set option, NULLCHARVAL= data set option

---

**Syntax**

```
NULLCHAR=SAS | YES | NO
```  

**Syntax Description**

- **SAS**
  - indicates that missing character values in SAS data sets are treated as NULL values if the DBMS allows these. Otherwise, they are treated as the NULLCHARVAL= value.

- **YES**
  - indicates that missing character values in SAS data sets are treated as NULL values if the DBMS allows these. Otherwise, an error is returned.

- **NO**
  - indicates that missing character values in SAS data sets are treated as the NULLCHARVAL= value—regardless of whether the DBMS allows NULL values for the column.

**Details**

This option affects insert and update processing. It also applies when you use DBINDEX= and DBKEY=.

It works with NULLCHARVAL=, which determines what is inserted when NULL values are not allowed. The DBMS treats all missing SAS numeric values (represented in SAS as '.') as NULL values.

**Oracle:** For interactions between NULLCHAR= and BULKLOAD=ZX’11, see the bulk-load topic in the Oracle section.
**NULLCHARVAL= Data Set Option**

Defines the character string that replaces missing SAS character values during insert, update, DBINDEX=, and DBKEY= processing.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** a blank character
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.
- **See:** DBFORCE= data set option, DBINDEX= data set option, DBKEY= data set option, DBNULL= data set option, NULLCHAR= data set option

### Syntax

`NULLCHARVAL='character-string'`

### Details

This option affects insert and update processing and also applies when you use DBINDEX= and DBKEY=.

It also works with NULLCHAR= to determine whether a missing SAS character value is treated as a NULL value. If NULLCHARVAL= is longer than the maximum column width, the string is truncated if DBFORCE=YES or the operation fails if DBFORCE=NO.

---

**OR_IDENTITY_COLS= Data Set Option**

specifies columns to be used to simulate an identity column.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Interaction:** The format of the user sequence for this option depends on the value of BULKLOAD.
- **Data source:** Oracle
- **Note:** Support for this data set option was added for SAS 9.4.
- **See:** BULKLOAD= data set option, BULKLOAD= LIBNAME option

### Syntax

`OR_IDENTITY_COLS=(column-name-1='user-sequence-1' <column-name-2='user-sequence-2' ... column-name-n='user-sequence-n'>)`
Syntax Description

column-name-N
specifies the name of an identity column.

user-sequence-N
specifies the expression that is used to generate the identity column values.

When BULKLOAD=YES, the user sequence should take the following form:
sequence(max,n)

When BULKLOAD=NO, the user sequence should take the following form:
<user-created-sequence>.nextval

Details

When BULKLOAD=YES, you supply the name of an identity column and the expression that is used to generate identity values. You supply the expression in the form sequence(max,n). This expression says that the values in the identity column are generated by taking the maximum identity value and adding the value n. For example, the expression sequence(max,2) increments generated identity values by 2.

When BULKLOAD=NO, you supply the name of an identity column and the expression that identifies the identity values. You supply the expression in the form user-created-sequence.nextval, such as t1_id.nextval.

OR_PARTITION= Data Set Option

Allows reading, updating, and deleting from a particular partition in a partitioned table, also inserting and bulk loading into a particular partition in a partitioned table.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Restriction: The partition name must be valid or an error occurs.
Data source: Oracle
Tip: This option is appropriate when reading, updating, and deleting from a partitioned table, also when inserting into a partitioned table or bulk loading to a table. You can use it to boost performance.

Syntax

OR_PARTITION= name of a partition in a partitioned Oracle table

Syntax Description

name of a partition in a partitioned Oracle table
specifies the partition name.

Details

Use this option in cases where you are working with only one particular partition at a time in a partitioned table. Specifying this option boosts performance because you are limiting your access to only one partition of a table instead of the entire table.
Examples

Example 1: Read, Update, Delete, Load, and Insert from a Partitioned Table

This example shows one way that you can use this option.

```
libname x oracle user=myusr1 pw=mypwd1 path=mypath;
proc datasets library=x;
    delete orparttest;run;
data x.ORparttest ( dbtype=(NUM='int')
    DBCREATE_TABLE_OPTS='partition by range (NUM)
        (partition p1 values less than (11),
         partition p2 values less than (21),
         partition p3 values less than (31),
         partition p4 values less than (41),
         partition p5 values less than (51),
         partition p6 values less than (61),
         partition p7 values less than (71),
         partition p8 values less than (81)
    )' );
do i=1 to 80;
    NUM=i;
    output;
end;
run;
options sastrace=",,t,d" sastraceloc=saslog nostsuffix;
/* input */
proc print data=x.orparttest ( or_partition=p4 );
run;
/* update */
proc sql;
/* update should fail with 14402, 00000, "updating partition key column
     would cause a partition change"
     // *Cause: An UPDATE statement attempted to change the value of a
     // partition key column causing migration of the row to
     // another partition.
     // *Action: Do not attempt to update a partition key column or make
     // sure that the new partition key is within the range
     // containing the old partition key.
     */
update x.orparttest ( or_partition=p4 ) set num=100;
update x.orparttest ( or_partition=p4 ) set num=35;
select * from x.orparttest ( or_partition=p4 );
select * from x.orparttest ( or_partition=p8 );
/* delete */
delete from x.orparttest ( or_partition=p4 );
select * from x.orparttest;
quit;
/* load to an existing table */
data new; do i=31 to 39; num=i; output;end;
run;
data new2; do i=1 to 9; num=i; output;end;
run;
proc append base= x.orparttest ( or_partition=p4 ) data= new;
run;
```
/* Insert should fail 14401, 00000, "inserted partition key is outside specified partition"
 // *Cause:   The concatenated partition key of an inserted record is outside the ranges of the two concatenated partition bound lists that delimit the partition named in the INSERT statement.
 // *Action: Do not insert the key or insert it in another partition.
 */
proc append base= x.orparttest ( or_partition=p4 ) data= new2;
run;
/* load to an existing table */
proc append base= x.orparttest ( or_partition=p4 bulkload=yes bl_load_method=truncate ) data= new;
run;
/* insert should fail 14401 */
proc append base= x.orparttest ( or_partition=p4 bulkload=yes bl_load_method=truncate ) data= new2;
run;

Example 2: Create and Manipulate a Partitioned Table

Here are a series of sample scenarios that illustrate how you can use this option. The first shows how to create the ORPARTTEST table, on which all remaining examples depend.

libname x oracle user=myusr1 pw=mypwd1 path=mypath;
proc datasets library=x;
delete orparttest;run;
data x.ORparttest ( dbtype=(NUM='int')
 DBCREATE_TABLE_OPTS='partition by range (NUM)
 (partition p1 values less than (11),
 partition p2 values less than (21),
 partition p3 values less than (31),
 partition p4 values less than (41),
 partition p5 values less than (51),
 partition p6 values less than (61),
 partition p7 values less than (71),
 partition p8 values less than (81))');
do i=1 to 80;
   NUM=i; output;
end;
run;

In this example, only the P4 partition is read.
proc print data=x.orparttest ( or_partition=p4 );
run;

Next, rows that belong to only the single P4 partition are updated.
proc sql;
update x.orparttest ( or_partition=p4 ) set num=35;
quit;

Although this code shows how a particular partition can be updated, updates and inserts to the partition key column can be done so that data must be migrated to a different partition in the table. This next example fails because the value 100 does not belong to the P4 partition.
proc sql;
update x.orparttest ( or_partition=p4 ) set num=100;

proc sql;
delete from x.orparttest ( or_partition=p4 );
quit;

Next, rows are added to the P4 partition in the table.

data new;
  do i=31 to 39; num=i; output;end;
run;
proc append base= x.orparttest ( or_partition=p4 );
  data= new;
run;

This example also adds rows to the P4 partition but uses the SQL*Loader instead.

proc append base= x.orparttest ( or_partition=p4 bulkload=yes );
  data= new;
run;

---

**OR_UPD_NOWHERE= Data Set Option**

Specifies whether SAS uses an extra WHERE clause when updating rows with no locking.

- Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- Alias: ORACLE_73_OR_ABOVE=
- Default: LIBNAME option setting
- Requirement: Due to the published Oracle bug 440366, sometimes an update on a row fails even if the row has not changed. Oracle offers this solution: When you create a table, increase the number of INITRANS to at least 3 for the table.
- Data source: Oracle
- See: Locking in the Oracle Interface, OR_UPD_NOWHERE= LIBNAME option, SASTRACE= system option, SASTRACELOC= system option, UPDATE_LOCK_TYPE= data set option

### Syntax

**OR_UPD_NOWHERE**=YES | NO

### Syntax Description

**YES**

specifies that SAS does not use an additional WHERE clause to determine whether each row has changed since it was read. Instead, SAS uses the SERIALIZABLE isolation level (available with Oracle7.3 and later) for update locking. If a row changes after the serializable transaction starts, the update on that row fails.

**NO**

specifies that SAS uses an additional WHERE clause to determine whether each row has changed since it was read. If a row has changed since being read, the update fails.
Details

Use this option when you are updating rows without locking (UPDATE_LOCK_TYPE=NOLOCK).

By default (OR_UPD_NOWHERE=YES), updates are performed in serializable transactions so that you can avoid problems with extra WHERE clause processing and potential WHERE clause floating-point precision.

Example: Create and Update a Table

In this example, you create a small Oracle table, TEST. You then update it once by using the default setting (OR_UPD_NOWHERE=YES) and once by specifying OR_UPD_NOWHERE=NO.

```sas
libname oralib oracle user=myusr1 pw=mypwd1 update_lock_type=no;
data oralib.test;
c1=1;
c2=2;
c3=3;
run;
options sastrace=",,,d" sastraceloc=saslog;
proc sql;
  update oralib.test set c2=22;
  update oralib.test(or_upd_nowhere=no) set c2=222;
quit;
```

This code uses the SASTRACE= and SASTRACELOC= system options to send the output to the SAS log.

---

**ORHINTS= Data Set Option**

Specifies Oracle hints to pass to Oracle from a SAS statement or SQL procedure.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Data source:** Oracle

**Syntax**

`ORHINTS='Oracle-hint'`

**Syntax Description**

- **Oracle-hint** specifies an Oracle hint for SAS/ACCESS to pass to the DBMS as part of an SQL query.

**Details**

If you specify an Oracle hint, SAS passes the hint to Oracle. If you omit ORHINTS=, SAS does not send any hints to Oracle.
Example: Pass a Hint

This example runs a SAS procedure on DBMS data and SAS converts the procedure to one or more SQL queries. ORHINTS= lets you specify an Oracle hint for SAS to pass as part of the SQL query.

libname mydblib oracle user=myusr1 password=mypwd1 path='myorapath';
proc print data=mydblib.payroll(orhints='/*+ ALL_ROWS */');
run;

In this example, SAS sends the Oracle hint '/*+ ALL_ROWS */' to Oracle as part of this statement:

SELECT /*+ ALL_ROWS */ * FROM PAYROLL

OVERLAPS= Data Set Option

determines whether the values for BUS_START and BUS_END can overlap between active records for the specified columns.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: 
Default: column names separated by commas
Interaction: allowed for the BUSINESS_TIME period
Data source: DB2 z/OS
See: TEMPORAL= data set option, BUSINESS_TIMEFRAME= data set option, SYSTEM_TIMEFRAME = data set option, BUSINESS_DATATYPE= data set option

Syntax

OVERLAPS='column names'

Syntax Description

column names
a list of column names, separated by commas, for which business time period overlap is not allowed.

Details

Here is an example that uses the OVERLAPS= option:

proc sql;
    create table policy (dbnull=(all_=no) temporal=business OVERLAPS='id,vin')
        {id,int,vin varchar(10)};
    <additional statements>
quit;
**PARMDEFAULT= Data Set Option**

Specifies whether the SAP HANA engine uses the defaults for variables and parameters that are specified in the metadata in SAP HANA.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Alias:** PARMDEFAULT, USE_PARAMETER_DEFAULT

**Data source:** SAP HANA

### Syntax

```
PARMDEFAULT=YES | NO
```

### Syntax Description

**YES**

specifies to look up metadata for variables and parameters. This enables you to apply defaults for variables in a WHERE clause, and defaults for input parameters using the PLACEHOLDER syntax.

**NO**

specifies to not look up metadata for variables and parameters.

### Details

The default for the PARMDEFAULT= option is YES if the PARMSTRING= LIBNAME or data set option is specified. The default is NO if no PARMSTRING= option is specified.

Applying the defaults requires additional queries to the metadata. It can be switched off to avoid making unnecessary queries.

---

**PARMSTRING= Data Set Option**

Specifies a quoted string of variable name and value pairs separated by a comma, or a placeholder string.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Alias:** PARAMETERS

**Data source:** SAP HANA

### Syntax

```
PARMSTRING=<"variable-name1=variable-value1,variable-name2=variable-value2,..."> < "'PLACEHOLDER' = (variable-name1, variable-value1), 'PLACEHOLDER' = (variable-name2, variable-value2),...">
```

---

*PARMSTRING= Data Set Option* 389
Syntax Description

"variable-name1=variable-value1"

specifies the variable name and value pair. More than one pair can be specified, and must be separated by a comma.

"PLACEHOLDER= ("variable-name1", "variable-value1")"

specifies the variable name and value pair as a placeholder.

Note: You can also combine a name and value pair and a placeholder:

PARMSTRING="variable-name1=variable-value1",
< 'PLACEHOLDER'= ("variable-name2", "variable-value2")"

Details

When you specify the variable name and value pairs, the SAS/ACCESS engine locates the variable input parameter in the SAP HANA metadata. The value is applied either as a WHERE clause for variables, or it generates and applies a PLACEHOLDER= string for passing the input parameters to the view execution. If the variable input parameter is not found in metadata, it is appended as part of a PLACEHOLDER= string to the generated SQL string.

If the user specifies a placeholder string, the string is passed directly to the SAP HANA query to be processed in SAP HANA.

Here are some syntax examples:

PARMSTRING = "parm_price=30"

PARMSTRING = "'PLACEHOLDER' = ("$$parm_product$$", 'Tablet')"

PARMSTRING = "PLACEHOLDER. $$parm_category$$' => 'Notebooks'"

When a PARMSTRING= LIBNAME option and a PARMSTRING= data set option are both specified, the PLACEHOLDER= string becomes a combination of both of the parameters and is passed to SAP HANA. An input parameter can occur only once as a placeholder in the SQL statement. If a parameter appears in the LIBNAME option and the data set option, the SAS/ACCESS engine tries to resolve this by passing only a fragment from the data set option.

You can access a view using its synonym. When accessing SAP HANA synonyms the metadata is located for the object.

Here is an example of the PARMSTRING= data set option:

data work.a;
  set a.PRODUCT_LIST_WITH_PARAM_DEF_S
  (parstring="parm_category='Notebooks'");
run;

Comparisons

The PARMSTRING= data set option is applied to the table or view that is specified.

The PARMSTRING= libname option is applied to all column tables and column views in the library. It will not be applied to row store objects.

PARTITION_KEY= Data Set Option

Specifies the column name to use as the partition key for creating fact tables.
**PARTITIONED_BY= Data Set Option**

Specifies the column name to use as the partition key for creating fact tables.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Requirements:</td>
<td>To create a data set in Aster without error, you must either set DIMENSION=YES (LIBNAME or data set option) or specify a partition key in the PARTITION_KEY= (LIBNAME or data set) option. You must enclose the column name in quotation marks.</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster</td>
</tr>
<tr>
<td>See:</td>
<td>DIMENSION= LIBNAME option, DIMENSION= data set option, PARTITION_KEY= LIBNAME option [contains examples]</td>
</tr>
</tbody>
</table>

### Syntax

**PARTITIONED_BY=**

### Details

Aster uses two table types, dimension and fact tables.

### PARTITIONED_BY= Data Set Option

Specifies the column name to use as the partition key for creating fact tables.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
</table>
| Aliases:  | PARTITION_BY
PARTITION |
| Default:  | none                                                                      |
| Requirements: | The SAS_HADOOP_RESTFUL environment variable must be set to 1 to use this data set option. You can set this option only when BULKLOAD is set to 1. |
| Data source: | Impala |
| Note: | Support for this data set option was added in the second maintenance release for SAS 9.4. |
| See: | PARTITIONED_BY= LIBNAME option (contains examples) |

### Syntax

**PARTITIONED_BY=**(column-name1<...column-nameN>)

**Required Argument**

*column-name1<...column-nameN>*

specifies the column names that are used to partition the table that is being bulk loaded. Separate multiple values with spaces and include the list in parentheses.

### Example:

```plaintext
data x.partitionsample(partitioned_by=(col4 col5) dbtype=(col4='int' col5='int'));```
This example shows how to load a table that is partitioned by columns Col4 and Col5.

### POST_STMT_OPTS= Data Set Option

Allows additional database-specific options to be placed after the CREATE TABLE statement in generated SQL code.

**Valid in:** DATA and PROC steps (when creating DBMS tables using SAS/ACCESS software).

**Alias:** DBCREATE_TABLE_OPTS=

**Default:** none

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for this option was added in the third maintenance release for SAS 9.4.

**See:** DBCREATE_TABLE_OPTS= data set option, DBIDIRECTEXEC system option, POST_TABLE_OPTS= data set option, PRE_STMT_OPTS= data set option, PRE_TABLE_OPTS= data set option

### Syntax

**POST_STMT_OPTS=**"DBMS-SQL-option(s)"

### Required Argument

**DBMS-SQL-option(s)**

specifies database-specific options to be placed after the CREATE TABLE statement. Enclose the options that you specify within single or double quotation marks.

### Details

You can use the POST_STMT_OPTS= data set option with these related data set options: PRE_STMT_OPTS, PRE_TABLE_OPTS=, and POST_TABLE_OPTS=. For example, you can supply database options according to this template:

```sql
proc sql;
create table crtab1 ( POST_TABLE_OPTS= /* post_table_hint */
  PRE_TABLE_OPTS= /* pre_table_hint */
  POST_STMT_OPTS= /* post_stmt_hint */
  PRE_STMT_OPTS= /* pre_stmt_hint */
) as
  select * from rdtab;
<additional-clauses>
quit;
```
The resulting code varies depending on whether the DBIDIRECTEXEC system option is enabled. When DBIDIRECTEXEC is set, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

```sql
/* pre_stmt_hint */ CREATE /* pre_table_hint */  TABLE DBID20.crtab1  
  /* post_table_hint */  as ( select TXT_1."C1", TXT_1."D1", TXT_1."E1"  
    from DBID20.RDTAB TXT_1 ) WITH NO DATA IN UDBID20.USERDATA  
  /* post_stmt_hint */
```

When you specify NODBIDIRECTEXEC, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

```sql
/* pre_stmt_hint */ CREATE /* pre_table_hint */  TABLE DBID20.CRTAB2  
  /* post_table_hint */ (C1 FLOAT, D1 CHAR(1), E1 CHAR(1)) IN UDBID20.USERDATA  
  /* post_stmt_hint */
```

Another usage template that does not use the SQL procedure is shown in this code:

```sas
data mylib.crtab1 ( POST_TABLE_OPTS= "/* post_table_hint */"
    PRE_TABLE_OPTS= "/* pre_table_hint */"
    POST_STMT_OPTS= "/* post_stmt_hint */"
    PRE_STMT_OPTS= "/* pre_stmt_hint */"
);
set work.localtable;
run;
```

---

**POST_TABLE_OPTS= Data Set Option**

Allows additional database-specific options to be placed after the table name in a CREATE TABLE statement.

**Valid in:** DATA and PROC steps (when creating DBMS tables using SAS/ACCESS software).

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, Netezza, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for this option was added in the third maintenance release for SAS 9.4.

**See:** DBIDIRECTEXEC system option, POST_STMT_OPTS= data set option, PRE_STMT_OPTS= data set option, PRE_TABLE_OPTS= data set option

---

**Syntax**

```
POST_TABLE_OPTS=DBMS-SQL-option(s)
```

**Required Argument**

**DBMS-SQL-option(s)** specifies additional database-specific options to be placed after the table name in a CREATE TABLE statement.

**Details**

You can use the POST_TABLE_OPTS= data set option with these related data set options: PRE_STMT_OPTS, POST_STMT_OPTS=, and PRE_TABLE_OPTS= data set option. For example, you can supply database options according to this template:
proc sql;
create table crtab1 ( /* post_table_hint */
POST_TABLE_OPTS= "/* post_table_hint */"
PRE_TABLE_OPTS=  "/* pre_table_hint */"
POST_STMT_OPTS=  "/* post_stmt_hint */"
PRE_STMT_OPTS=   "/* pre_stmt_hint */"
) as
  select * from rdtab;
<additional-clauses>
quit;

The resulting code varies depending on whether the DBIDIRECTEXEC system option is enabled. When DBIDIRECTEXEC is set, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

/* pre_stmt_hint */ CREATE  /* pre_table_hint */  TABLE DBID20.crtab1
/* post_table_hint */
/* post_stmt_hint */
(C1 FLOAT, D1 CHAR(1), E1 CHAR(1)) IN UDBID20.USERDATA
When you specify NODBIDIRECTEXEC, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

/* pre_stmt_hint */ CREATE  /* pre_table_hint */  TABLE DBID20.CRTAB2
/* post_table_hint */
/* post_stmt_hint */
(C1 FLOAT, D1 CHAR(1), E1 CHAR(1)) IN UDBID20.USERDATA

Another usage template that does not use the SQL procedure is shown in this code:

data mylib.crtab1 ( POST_TABLE_OPTS= "/* post_table_hint */"
PRE_TABLE_OPTS= "/* pre_table_hint */"
POST_STMT_OPTS= "/* post_stmt_hint */"
PRE_STMT_OPTS= "/* pre_stmt_hint */"
);
set work.localtable;
run;

---

**PRE_STMT_OPTS= Data Set Option**

Allows additional database-specific options to be placed before a CREATE TABLE statement.

**Valid in:** DATA and PROC steps (when creating DBMS tables using SAS/ACCESS software).

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica

**Note:** Support for this option was added in the third maintenance release for SAS 9.4.

**See:** DBIDIRECTEXEC system option, POST_STMT_OPTS= data set option,
POST_TABLE_OPTS= data set option, PRE_TABLE_OPTS= data set option

**Syntax**

`PRE_STMT_OPTS=DBMS-SQL-option(s)`
**Required Argument**

**DBMS-SQL-option(s)**

specifies additional database-specific options to be placed before the CREATE TABLE statement.

**Details**

You can use the `PRE_STMT_OPTS=` data set option with these related data set options: `PRE_TABLE_OPTS=`, `POST_STMT_OPTS=`, and `POST_TABLE_OPTS=`. For example, you can supply database options according to this template:

```sql
proc sql;
create table crtab1 ( POST_TABLE_OPTS= "/* post_table_hint */"
  PRE_TABLE_OPTS= "/* pre_table_hint */"
  POST_STMT_OPTS= "/* post_stmt_hint */"
  PRE_STMT_OPTS= "/* pre_stmt_hint */"
) as
  select * from rdtab;
<additional-clauses>
quit;
```

The resulting code varies depending on whether the `DBIDIRECTEXEC` system option is enabled. When `DBIDIRECTEXEC` is set, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

```sql
/* pre_stmt_hint */ CREATE /* pre_table_hint */ TABLE DBID20.crtab1
/* post_table_hint */ as ( select TXT_1."C1", TXT_1."D1", TXT_1."E1"
  from DBID20.RDTAB TXT_1 ) WITH NO DATA IN UDBID20.USERDATA
/* post_stmt_hint */
```

When you specify `NODBIDIRECTEXEC`, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

```sql
/* pre_stmt_hint */ CREATE /* pre_table_hint */ TABLE DBID20.CRTAB2
/* post_table_hint */ (C1 FLOAT, D1 CHAR(1), E1 CHAR(1)) IN UDBID20.USERDATA
/* post_stmt_hint */
```

Another usage template that does not use the SQL procedure is shown in this code:

```sas
data mylib.crtab1 ( POST_TABLE_OPTS= "/* post_table_hint */"
  PRE_TABLE_OPTS= "/* pre_table_hint */"
  POST_STMT_OPTS= "/* post_stmt_hint */"
  PRE_STMT_OPTS= "/* pre_stmt_hint */"
);
set work.localtable;
run;
```

---

**PRE_TABLE_OPTS= Data Set Option**

allows additional database-specific options to be placed before the table name in a CREATE TABLE statement.

**Valid in:** DATA and PROC steps (when creating DBMS tables using SAS/ACCESS software).

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica
Support for this option was added in the third maintenance release for SAS 9.4.

**See:** DBIDIRECTEXEC system option, POST_STMT_OPTS= data set option, POST_TABLE_OPTS= data set option, PRE_STMT_OPTS= data set option

**Syntax**

PRE_TABLE_OPTS=**DBMS-SQL-option(s)**

**Required Argument**

**DBMS-SQL-option(s)**

specifies additional database-specific options to be placed before the table name in a CREATE TABLE statement.

**Details**

You can use the PRE_TABLE_OPTS= data set option with these related data set options: PRE_STMT_OPTS, POST_STMT_OPTS=, and POST_TABLE_OPTS=. For example, you can supply database options according to this template:

```
proc sql;
create table ctab1 ( POST_TABLE_OPTS= "/* post_table_hint */"
                     PRE_TABLE_OPTS= "/* pre_table_hint */"
                     POST_STMT_OPTS= "/* post_stmt_hint */"
                     PRE_STMT_OPTS= "/* pre_stmt_hint */"
                ) as
   select * from rdtab;
<additional-clauses>
quit;
```

The resulting code varies depending on whether the DBIDIRECTEXEC system option is enabled. When DBIDIRECTEXEC is set, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

```
/* pre_stmt_hint */ CREATE /* pre_table_hint */ TABLE DBID20.crtab1
/* post_table_hint */ as ( select TXT_1."C1", TXT_1."D1", TXT_1."E1"
   from DBID20.RDTAB TXT_1 ) WITH NO DATA IN UDBID20.USERDATA
/* post_stmt_hint */
```

When you specify NODBIDIRECTEXEC, the code might result in this generated SQL code (assuming the columns C1, D1, and E1):

```
/* pre_stmt_hint */ CREATE /* pre_table_hint */ TABLE DBID20.CRTAB2
/* post_table_hint */ (C1 FLOAT, D1 CHAR(1), E1 CHAR(1)) IN UDBID20.USERDATA
/* post_stmt_hint */
```

Another usage template that does not use the SQL procedure is shown in this code:

```
data mylib.crtab1 ( POST_TABLE_OPTS= "/* post_table_hint */"
                   PRE_TABLE_OPTS= "/* pre_table_hint */"
                   POST_STMT_OPTS= "/* post_stmt_hint */"
                   PRE_STMT_OPTS= "/* pre_stmt_hint */"
                 );
set work.localtable;
run;
```
PRESERVE_COL_NAMES= Data Set Option

Preserves spaces, special characters, and case sensitivity in DBMS column names when you create DBMS tables.

Valid in: DATA and PROC steps (when creating DBMS tables using SAS/ACCESS software).

Alias: PRESERVE_NAMES= (see "Details")

Default: LIBNAME option setting

Interaction: If you use the DS2 or FedSQL language, quoting and casing of names is different. For more information, see the identifiers topic in SAS DS2 Language Reference or SAS FedSQL Language Reference.

Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: PRESERVE_COL_NAMES= LIBNAME option, PRESERVE_TAB_NAMES= LIBNAME option, VALIDVARNAMES= system option, SAS Names and Support for DBMS Names, and the DBMS-specific naming conventions sections for your SAS/ACCESS interface

Syntax

PRESERVE_COL_NAMES= YES | NO

Syntax Description

NO

specifies that column names that are used in DBMS table creation are derived from SAS variable names by using the SAS variable name normalization rules. (For more information see the VALIDVARNAMES= system option.) However, the database applies its DBMS-specific normalization rules to the SAS variable names when it creates the DBMS column names.

The use of name literals to create column names that use database keywords or special symbols other than the underscore character might be invalid when DBMS normalization rules are applied. To include nonstandard SAS symbols or database keywords, specify PRESERVE_COL_NAMES=YES.

YES

specifies that column names that are used in table creation are passed to the DBMS with special characters and the exact, case-sensitive spelling of the name preserved.

Details

This option applies only when you use SAS/ACCESS to create a new DBMS table. When you create a table, you assign the column names by using one of these methods:

• To control the case of the DBMS column names, specify variables with the desired case and set PRESERVE_COL_NAMES=YES. If you use special symbols or blanks, you must set VALIDVARNAMES=ANY and use name literals. For more information, see the naming topic in this document and also SAS Data Set Options: Reference.
**SAP HANA**: When you specify `PRESERVE_COL_NAMES=YES`, you can use reserved words for column names.

- So that the DBMS to normalize the column names according to its naming conventions, specify variables with any case and set `PRESERVE_COL_NAMES=NO`.

When you use SAS/ACCESS to read from, insert rows into, or modify data in an existing DBMS table, SAS identifies the database column names by their spelling. Therefore, when the database column exists, the case of the variable does not matter.

For more information, see the SAS/ACCESS naming topic in the DBMS-specific reference section for your interface.

To save some time when coding, specify the `PRESERVE_NAMES= alias if you plan to specify both `PRESERVE_COL_NAMES= and `PRESERVE_TAB_NAMES=` LIBNAME options.

To use column names in your SAS program that are not valid SAS names, you must use one of these techniques.

- Use the `DQUOTE=` option in PROC SQL and reference your columns using double quotation marks. Here is an example.

  ```
  proc sql dquote=ansi;
  select "Total$Cost" from mydblib.mytable;
  ```

- Specify the global `VALIDVARNAME=ANY` system option and use name literals in the SAS language. Here is an example.

  ```
  proc print data=mydblib.mytable;
  format 'Total$Cost'n 22.2;
  ```

If you are creating a table in PROC SQL, you must also include the `PRESERVE_COL_NAMES=YES` option. Here is an example.

```
libname mydblib oracle user=myusr1 password=mypwd1;
proc sql dquote=ansi;
  create table mydblib.mytable (preserve_col_names=yes) ("my$column" int);
```

`PRESERVE_COL_NAMES= does not apply to the pass-through facility.

---

**QUALIFIER= Data Set Option**

Specifies the qualifier to use when you are reading database objects, such as DBMS tables and views.

- Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- Default: LIBNAME option setting
- Data source: Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, PostgreSQL, Vertica
- See: QUALIFIER= LIBNAME option

**Syntax**

```
QUALIFIER=<qualifier-name>
```
Details

If this option is omitted, the default qualifier name, if any, is used for the data source. QUALIFIER= can be used for any data source, such as a DBMS object, that allows three-part identifier names: qualifier.schema.object.

QUERY_BAND= Data Set Option

Specifies whether to set a query band for the current transaction.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Data source:</td>
<td>Teradata</td>
</tr>
</tbody>
</table>
| See: | BULKLOAD= LIBNAME option  
BULKLOAD= data set option  
FASTEXPORT= LIBNAME option  
Maximizing Teradata Load Performance  
MULTILOAD= data set option  
QUERY_BAND= LIBNAME option |

Syntax

QUERY_BAND="pair-name=pair_value;"

Syntax Description

"pair-name=pair_value;"
  specifies a name and value pair of a query band for the current transaction.

Details

Use this option to set unique identifiers on Teradata transactions and to add them to the current transaction. The Teradata engine uses this syntax to pass the name-value pair to Teradata:

data db.new_table (QUERY_BAND="org=Sales; process=Build_Initial;*");  
set work.sasdata1;  
run;

For more information about this option and query-band limitations, see Teradata SQL Reference: Data Definition Statements.

QUERY_TIMEOUT= Data Set Option

Specifies the number of seconds of inactivity to wait before canceling a query.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>Data source:</td>
<td>Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, PostgreSQL, SAP HANA, Sybase IQ, Vertica</td>
</tr>
</tbody>
</table>
Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: `QUERY_TIMEOUT=` LIBNAME option

Syntax

```
QUERY_TIMEOUT=number-of-seconds
```

Details

`QUERY_TIMEOUT=0` indicates that there is no time limit for a query. This option is useful when you are testing a query, you suspect that a query might contain an endless loop, or the data is locked by another user.

---

**READBUFF= Data Set Option**

Specifies the number of rows of DBMS data to read into the buffer.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Aliases:** `ROWSET_SIZE= [DB2 under UNIX and PC Hosts, DB2 under z/OS, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, Sybase, Sybase IQ]`

**ROWSET= [Sybase IQ]**

**Default:** LIBNAME option setting

**Restriction:** When `READBUFF=1`, only one row is retrieved at a time.

**Interaction:** Buffering data reads can decrease network activities and increase performance. However, because SAS stores the rows in memory, higher values for `READBUFF=` use more memory. In addition, if too many rows are selected at once, rows that are returned to the SAS application might be out of date. For example, if someone else modifies the rows, you do not see the changes.

**Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica

**Note:** Support for HAWQ was added in the third maintenance release for SAS 9.4.

**Tips:** This option improves performance by specifying a number of rows that can be held in memory for input into SAS.

The higher the value for `READBUFF=`, the more rows that the engine retrieves in one fetch operation.

**See:** `INSERTBUFF=` LIBNAME option, `INSERTBUFF=` data set option, `READBUFF=` LIBNAME option

---

**Syntax**

```
READBUFF=integer
```

**Syntax Description**

`integer` is the positive number of rows to hold in memory. SAS allows the maximum number that the DBMS allows.
READ_ISOLATION_LEVEL= Data Set Option

Specifies which level of read isolation locking to use when you are reading data.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: LIBNAME option setting

Interaction: This option is ignored if you do not set READ_LOCK_TYPE=ROW. [DB2 under UNIX and PC Hosts, Netezza, ODBC]

Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, OLE DB, Oracle, Sybase, Teradata, Vertica

See: READ_ISOLATION_LEVEL= LIBNAME option, READ_LOCK_TYPE= LIBNAME option, READ_LOCK_TYPE= data set option, UPDATE_ISOLATION_LEVEL= LIBNAME option, UPDATE_ISOLATION_LEVEL= data set option, UPDATE_LOCK_TYPE= LIBNAME option, UPDATE_LOCK_TYPE= data set option, and DBMS-specific locking information in the reference section for your SAS/ACCESS interface

Syntax

READ_ISOLATION_LEVEL=DBMS-specific-value

Syntax Description

dbms-specific-value
See the DBMS-specific reference section for your interface for this value.

Details

The degree of isolation defines the degree to which these items are affected:

• Rows that the current application reads and updates are available to other concurrently executing applications.

• Update activity of other concurrently executing application processes can affect the current application.

READ_LOCK_TYPE= Data Set Option

Specifies how data in a DBMS table is locked during a read transaction.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: LIBNAME option setting

Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, OLE DB, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica

Tip: If you omit READ_LOCK_TYPE=, you receive either the default action for the DBMS that you are using or a lock for the DBMS that was set with the LIBNAME statement.

See: READ_ISOLATION_LEVEL= LIBNAME option, READ_ISOLATION_LEVEL= data set option, READ_LOCK_TYPE= LIBNAME option, UPDATE_ISOLATION_LEVEL= LIBNAME option, UPDATE_ISOLATION_LEVEL= data set option,
UPDATE_LOCK_TYPE= LIBNAME option, UPDATE_LOCK_TYPE= data set option, and DBMS-specific locking information in the reference section for your SAS/ACCESS interface.

Syntax

READ_LOCK_TYPE= ROW | PAGE | TABLE | NOLOCK | VIEW

Syntax Description

ROW
locks a row if any of its columns are accessed. If you are using the interface to ODBC or DB2 under UNIX and PC Hosts, READ_LOCK_TYPE=ROW indicates that locking is based on the READ_ISOLATION_LEVEL= LIBNAME option.

Data source: DB2 under UNIX and PC Hosts, Microsoft SQL Server, ODBC, Oracle, PostgreSQL, Sybase IQ, Vertica

PAGE
locks a page of data, which is a DBMS-specific number of bytes.

Data source: Sybase

TABLE
locks the entire DBMS table. If you specify READ_LOCK_TYPE=TABLE, you must also specify the CONNECTION=UNIQUE, or you receive an error message. Setting CONNECTION=UNIQUE ensures that your table lock is not lost (for example, due to another table closing and committing rows in the same connection).

Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, Oracle, Sybase IQ, Teradata

NOLOCK
does not lock the DBMS table, pages, or any rows during a read transaction.

Data source: Microsoft SQL Server, ODBC with Microsoft SQL Server driver, OLE DB, Oracle, Sybase

VIEW
locks the entire DBMS view.

Data source: Teradata

READ_METHOD= Data Set Option

Specifies how to read data.

Valid in: DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

Default: none

Data source: Hadoop

See: READ_METHOD= LIBNAME option
Syntax

READ_METHOD=JDBC | HDFS

Syntax Description

JDBC
specifies that data is to be read through the JDBC connection to the Hive service.

HDFS
specifies that data is to be read through a connection to the Hadoop HDFS service.

Details

Although HDFS cannot alter the behavior of operations that always use JDBC, in general HDFS is a faster alternative to JDBC. To take advantage of potential performance benefits, set this option to HDFS. Use JDBC when you cannot access the HDFS service or JDBC Read offers some other advantage.

Example: Read Data Using JDBC

In this example, a partition of data from the sales Hive table is read using JDBC.

libname hdp hadoop server=mysrv1 user=myusr1 pwd=mypwd1;
data work.sales_subset; set hdp.sales(READ_METHOD=JDBC);
where year_month='2012-10'; run;

READ_MODE_WAIT= Data Set Option

Specifies during SAS/ACCESS Read operations whether Teradata waits to acquire a lock or fails your request when a different user has locked the DBMS resource.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: LIBNAME option setting
Data source: Teradata
See: READ_MODE_WAIT= LIBNAME option, Locking in the Teradata Interface

Syntax

READ_MODE_WAIT=YES | NO

Syntax Description

YES
specifies that Teradata waits to acquire the lock, and SAS/ACCESS waits indefinitely until it can acquire the lock.

NO
specifies that Teradata fails the lock request if the specified DBMS resource is locked.
Details

If you specify READ_MODE_WAIT=NO, and a different user holds a restrictive lock, then the executing SAS step fails. SAS/ACCESS continues to process the job by executing the next step. If you specify READ_MODE_WAIT=YES, SAS/ACCESS waits indefinitely until it can acquire the lock.

A restrictive lock means that another user is holding a lock that prevents you from obtaining your desired lock. Until the other user releases the restrictive lock, you cannot obtain your lock. For example, another user's table-level WRITE lock prevents you from obtaining a READ lock on the table.

For more information, see locking topic in the Teradata section.

ROW_DELIMITER= Data Set Option

Specifies the single delimiter character to use to separate Hive table records.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Aliases: ROW_DELIM
ROW_DELIMIT

Default: 1010 (\n)

Restriction: Hive rejects this delimiter if you specify it in any other way, such as ROW_DELIMITER=010 (=linefeed).

Requirement: Specify a single-character or three-digit decimal ASCII value between 001 and 127.

Data source: Hadoop

See: COLUMN_DELIMITER= data set option

Examples: ROW_DELIMITER=#
ROW_DELIMITER=009 (tab)

Syntax

ROW_DELIMITER=single-character

SASDATEFMT= Data Set Option

Changes the SAS date format of a DBMS column.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: DBMS-specific

Data source: Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica

Note: Support for HAWQ was added in the third maintenance release for SAS 9.4.

See: DBSASTYPE= data set option
Syntax

\[
\text{SASDATEFMT=} ( \text{DBMS-date-col-1} = \text{SAS-date-format} \\
< \ldots \text{DBMS-date-col-n} = \text{SAS-date-format} > )
\]

**Syntax Description**

**DBMS-date-col**
- Specifies the name of a date column in a DBMS table.

**SAS-date-format**
- Specifies a SAS date format that has an equivalent (like-named) informat. For example, DATETIME21.2 is both a SAS format and a SAS informat, so it is a valid value for the \text{SAS-date-format} argument.

**Details**

If the SAS column date format does not match the date format of the corresponding DBMS column, convert the SAS date values to the appropriate DBMS date values. Use the SASDATEFMT= option to convert date values from the default SAS date format to another SAS date format that you specify.

Use the SASDATEFMT= option to prevent date type mismatches in these circumstances:

- during input operations to convert DBMS date values to the correct SAS DATE, TIME, or DATETIME values
- during output operations to convert SAS DATE, TIME, or DATETIME values to the correct DBMS date values.

The column names specified in this option must be DATE, DATETIME, or TIME columns; columns of any other type are ignored.

The format specified must be a valid date format; output with any other format is unpredictable.

If the SAS date format and the DBMS date format match, this option is not needed.

The default SAS date format is DBMS-specific and is determined by the data type of the DBMS column. See the DBMS-specific reference section for your SAS/ACCESS interface.

**Note:** For non-English date types, SAS automatically converts the data to the SAS type of NUMBER. SASDATEFMT= does not currently handle these date types. However, you can use a PROC SQL view to convert the DBMS data to a SAS date format as you retrieve the data, or use a format statement in other contexts.

**Oracle:** It is recommended that you use DBSASTYPE= instead of SASDATEFMT=.

**Examples**

**Example 1: Change the Date Format in Oracle**

In this example, the APPEND procedure adds SAS data from the SASLIB.DELAY data set to the Oracle table that is accessed by MYDBLIB/internat. Using SASDATEFMT=, the default SAS format for the Oracle column DATES is changed to the DATE9. format. Data output from SASLIB.DELAY into the DATES column in MYDBLIB/internat now converts from the DATE9. format to the Oracle format assigned to that type.
libname mydblib oracle user=myusr1 password=mypwd1;
libname saslib 'your-SAS-library';
proc append base=mydblib.internat(sasdatefmt=(dates='date9.'))force
data=saslib.delay;
run;

Example 2: Change a SAS Date Format to a Teradata Format
In the next example, SASDATEFMT= converts DATE1, a SAS DATETIME value, to a Teradata date column named DATE1.

libname x teradata user=myusr1 password=mypwd1;
proc sql noerrorstop;
create table x.dateinfo ( date1 date );
insert into x.dateinfo
(sasdatefmt=( date1='datetime21.' ) )
values ( '31dec2000:01:02:30'dt );
run;

Example 3: Change a Teradata Date Format to a SAS Format
In this example, SASDATEFMT= converts DATE1, a Teradata date column, to a SAS DATETIME type named DATE1.

libname x teradata user=myusr1 password=mypwd1;
data sas_local;
format date1 datetime21.;
set x.dateinfo( sasdatefmt=( date1='datetime21.' ) );
run;

SCHEMA= Data Set Option
Allows reading of such database objects as tables and views in the specified schema.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Alias: DATABASE= [Impala, Teradata]
Defaults: LIBNAME option setting [Aster, DB2 under UNIX and PC Hosts, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Vertica]
AUTHID= [DB2 under z/OS]
Data source: Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Impala, Informix, Microsoft SQL Server, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
Notes: Support for Netezza was added in the second maintenance release for SAS 9.4.
Support for HAWQ was added in the third maintenance release for SAS 9.4.
See: PRESERVE_TAB_NAMES= LIBNAME option, SCHEMA= LIBNAME option

Syntax
SCHEMA=schema-name
**Syntax Description**

`schema-name`

specifies the name that is assigned to a logical classification of objects in a relational database.

**Details**

For this option to work, you must have appropriate privileges to the schema that is specified.

If you do not specify this option, you connect to the default schema for your DBMS.

The values for SCHEMA= are usually case sensitive, so be careful when you specify this option.

*Aster:* The default is **none**, which uses the database user's default schema. When the user's default schema is the user name, the user name is used instead. An example is when SQLTables is called to obtain a table listing using PROC DATASETS or SAS Explorer.

*Netezza:* Starting in Netezza 7.0.3, you can configure your Netezza server for multiple schema support. The Netezza System Administrator’s Guide describes how to provision schema support. Multiple schema support is not enabled by default. The default is the database user’s default schema.

*Informix:* The SCHEMA data set option disables implicit pass-through.

*Oracle:* The default is the LIBNAME setting. If PRESERVE_TAB_NAMES=NO, SAS converts the SCHEMA= value to uppercase because all values in the Oracle data dictionary are converted to uppercase unless quoted.

*Sybase:* You cannot use the SCHEMA= option when you use UPDATE_LOCK_TYPE=PAGE to update a table.

*Teradata:* The default is the LIBNAME setting. If you omit this option, a libref points to your default Teradata database, which often has the same name as your user name. You can use this option to point to a different database. This option lets you view or modify a different user's DBMS tables or views if you have the required Teradata privileges. (For example, to read another user's tables, you must have the Teradata privilege SELECT for that user's tables.) For more information about changing the default database, see the DATABASE statement in your Teradata documentation.

**Example**

In this example, SCHEMA= causes DB2 to interpret MYDB.TEMP_EMPS as SCOTT.TEMP_EMPS.

```sas
proc print data=mydb.temp_emps
   schema=SCOTT;
run;
```

In this next example, SAS sends any reference to Employees as Scott.Employees.

```sas
libname mydblib oracle user=myusr1 password=mypwd1 path="myorapath";
proc print data=employees (schema=scott);
run;
```

In this example, user MYUSR1 prints the contents of the Employees table, which is located in the Donna database.

```sas
libname mydblib teradata user=myusr1 pw=mypwd1;
```
proc print data=mydblib.employees(schema=donna);
run;

**SEGMENT_NAME= Data Set Option**

Lets you control the segment in which you create a table.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** none
- **Data source:** Sybase

**Syntax**

```
SEGMENT_NAME=segment-name
```

**Syntax Description**

`segment-name`

specifies the name of the segment in which to create a table.

**SESSIONS= Data Set Option**

Specifies how many Teradata sessions to be logged on when using FastLoad, FastExport, or MultiLoad.

- **Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)
- **Default:** none
- **Data source:** Teradata
- **See:** SESSIONS= LIBNAME option

**Syntax**

```
SESSIONS=number-of-sessions
```

**Syntax Description**

`number-of-sessions`

specifies a numeric value that indicates the number of sessions to be logged on.

**Details**

When reading data with FastExport or loading data with FastLoad and MultiLoad, you can request multiple sessions to increase throughput. Using large values might not necessarily increase throughput due to the overhead associated with session management. Check whether your site has any recommended value for the number of sessions to use. See your Teradata documentation for details about using multiple sessions.
Example: Request Sessions to Load a Table

This example uses SESSIONS= in a LIBNAME statement to request that five sessions be used to load data with FastLoad.

```
libname x teradata user=myusr1 pw=mypwd1;
proc datasets library=x;
  delete test;run;
data x.test(FASTLOAD=YES SESSIONS=5);
i=5;
run;
```

SET= Data Set Option

Specifies whether duplicate rows are allowed when creating a table.

- **Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)
- **Alias:** TBLSET
- **Default:** NO
- **Data source:** Teradata

**Syntax**

```
SET= YES | NO
```

**Syntax Description**

- **YES**
  - specifies that no duplicate rows are allowed.
- **NO**
  - specifies that duplicate rows are allowed.

**Details**

Use the SET= data set option to specify whether to allow duplicate rows when creating a table. This option overrides the default Teradata MULTISET characteristic.

**Example: Create a Table without Duplicate Rows**

This example creates a Teradata table of type SET that does not allow duplicate rows.

```
libname trlib teradata user=myusr1 pw=mypwd1;
options sastrace=',,d' sastraceloc=saslog;
proc datasets library=x;
  delete test1;run;
data x.test1(TBLSET=YES);
i=1;output;
run;
```
SLEEP= Data Set Option

Specifies the number of minutes that FastExport, FastLoad, or MultiLoad waits before trying again to log on to Teradata.

Valid in: DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)

Default: 6

Data source: Teradata

Tip: The data set option has precedence over the LIBNAME option.

See: DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option, DBSLICEPARM= system option, FASTEXPORT= LIBNAME option, Maximizing Teradata Load Performance, MULTILOAD= data set option, SLEEP= LIBNAME option, TENACITY= LIBNAME option, TENACITY= data set option, Using the TPT API

Syntax

SLEEP=number-of-minutes

Syntax Description

number-of-minutes

the number of minutes to wait before trying again to log on to Teradata.

Details

Use this option to indicate to FastExport, FastLoad, or MultiLoad how long to wait before it retries logging on to Teradata when the maximum number of utilities are already running. (The maximum number of Teradata utilities that can run concurrently varies from 5 to 15, depending on the database server setting.) The default value for SLEEP= is 6 minutes. The value that you specify for SLEEP= must be greater than 0.

Use SLEEP= with TENACITY=, which specifies the time in hours that FastExport, FastLoad, or MultiLoad must continue to try the logon operation. SLEEP= and TENACITY= function very much like the SLEEP and TENACITY run-time options of the native Teradata FastExport, FastLoad, or MultiLoad utility.

SYSTEM_TIMEFRAME= Data Set Option

lets you provide a datetime range to use when querying or modifying a temporal table.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: none

Data source: DB2 for z/OS

See: TEMPORAL= data set option, OVERLAPS= data set option,BUSINESS_DATATYPE= data set option,, BUSINESS_TIMEFRAME= data set option
**Syntax**

SYSTEM_TIMEFRAME=FROM {datetime} TO {datetime2}

**Syntax Description**

{datetimeN} specifies a datetime value that defines the beginning or end of a time period that is used when you query or modify a table that contains temporal data.

Provide datetime values that include year, month, day, hours, minutes, seconds, and fractions of a second. For example, use the following code to specify the time period from midnight, November 1, 2012 to midnight, December 1, 2030:

```
system_timeframe="from '2012-11-01-00.00.00.0' to '2030-12-01-00.00.00.0'"
```

**TABLE_TYPE= Data Set Option**

Specifies the type of temporary tables and table storage when the engine creates tables in SAP HANA.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Access
- **Default:** none
- **Restrictions:**
  - If you do not specify a value for this option, tables that are created in SAP HANA follow the SAP HANA default for row or column store.
  - ROW and COLUMN are mutually exclusive.
  - LOCAL | LOCAL TEMPORARY and GLOBAL | GLOBAL TEMPORARY are mutually exclusive.
  - Do not specify LOCAL | LOCAL TEMPORARY or GLOBAL | GLOBAL TEMPORARY for permanent tables.
- **Interaction:** The TABLE_TYPE data set option overrides the TABLE_TYPE LIBNAME option.
- **Data source:** SAP HANA
- **See:** CONNECTION= (for sharing sessions across librefs), TABLE_TYPE= LIBNAME option
- **Examples:**
  - TABLE_TYPE=ROW
  - TABLE_TYPE=COLUMN
  - TABLE_TYPE=LOCAL
  - TABLE_TYPE=GLOBAL
  - TABLE_TYPE=(COLUMN GLOBAL)

**Syntax**

TABLE_TYPE=ROW | COLUMN | LOCAL | LOCAL TEMPORARY | GLOBAL | GLOBAL TEMPORARY

**Syntax Description**

ROW

creates a table using ROW-based storage in SAP HANA.
COLUMN
creates a table using COLUMN-based storage in SAP HANA.

LOCAL | LOCAL TEMPORARY
creates a local temporary table in SAP HANA. The table definition and data are visible only in the current session.

GLOBAL | GLOBAL TEMPORARY
creates a global temporary table in SAP HANA. The global temporary tables are globally available, and the data is visible only in the current session.

Details
This option takes effect when a SAS program creates a table in SAP HANA.

**TEMPORAL= Data Set Option**

specifies the type of temporal data that is contained in a table.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software) |
| Default: | none |
| Requirement: | DB2 z/OS with SAS 9.4 |
| Data source: | DB2 for z/OS |

See: OVERLAPS= data set option, BUSINESS_DATATYPE= data set option, BUSINESS_TIMEFRAME= data set option, SYSTEM_TIMEFRAME= data set option, "Temporal Data for DB2 under z/OS" on page 573

**Syntax**

TEMPORAL=BUSINESS | SYSTEM | BITEMPORAL

**Syntax Description**

BUSINESS
specifies a temporal table that contains business time. The BUSINESS options add the following columns to the end for storing temporal data: BUS_START and BUS_END.

SYSTEM
specifies a temporal table that contains system time. The SYSTEM option adds the following columns to the table for storing temporal data: SYS_START, SYS_END, and TRANS_START.

BITEMPORAL
specifies a temporal table that contains business and system time data. The BITEMPORAL option adds the following columns to the table for storing temporal data: SYS_START, SYS_END, BUS_START, BUS_END, and TRANS_START.

**TEMPORAL_QUALIFIER= Data Set Option**

Specifies time-dimension criteria for retrieving data from Teradata.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software) |
Default: CURRENT VALIDTIME [valid-time column], CURRENT TRANSACTIONTIME [transaction-time column]

Interaction: Specifying values in a DATA step overrides LIBNAME values.

Data source: Teradata

See: TEMPORAL_QUALIFIER= LIBNAME option

Syntax:

TEMPORAL_QUALIFIER= CURRENT VALIDTIME
   | 'VALIDTIME AS OF PERIOD 'period' | SEQUENCED VALIDTIME
   | 'NONSEQUENCED VALIDTIME " period"'
TEMPORAL_QUALIFIER= CURRENT TRANSACTIONTIME
   | 'TRANSACTIONTIME AS OF PERIOD 'period'
   | NONSEQUENCED TRANSACTIONTIME

Syntax Description

CURRENT VALIDTIME
   selects rows that are valid at the current time.

VALIDTIME AS OF PERIOD 'period'
   selects rows with valid-time periods that overlap the specified AS OF period. For the period, you can specify either a single date or a time period (date range) by specifying a start date and an end date.

SEQUENCED VALIDTIME PERIOD 'period'
   selects history, current, or future rows that are valid for the specified time period.

NONSEQUENCED VALIDTIME PERIOD 'period'
   treats the table as nontemporal.

CURRENT TRANSACTIONTIME
   selects rows that are open in transaction time.

TRANSACTIONTIME AS OF 'period'
   selects rows with transaction-time periods that overlap the specified AS OF period. For the period, you can specify either a single date or a time period (date range) by specifying a start date and an end date.

NONSEQUENCED TRANSACTIONTIME PERIOD 'period'
   treats the table as nontemporal.

Details

Use temporal qualifiers to specify time criteria for selecting data from temporal tables.

To use them, before the SQL add a value that you specify for one or more temporal qualifiers for a data set. For example, if you specify TEMPORAL_QUALIFIER='AS OF PERIOD ' (1999-01-01, 2099-01-05)' ' in a DATA step, 'AS OF PERIOD ' (1999-01-01, 2099-01-05)' ' is added before the SQL to select the data.

Examples

Example 1: Select Valid-Time Rows at the Current Time

/* Consider data as of 1995-01-01. */
libname x teradata user=myusr1 pw=mypwd1 server=mysrv1
  TEMPORAL_QUALIFIER='VALIDTIME AS OF DATE '1995-01-01' '

/* ASOF PERIOD '1999-01-01, 2099-01-05)' select * from mytest is submitted. */
proc print data=x.mytest(TEMPORAL_QUALIFIER='CURRENT VALIDTIME');
run;

**Example 2: Select Valid-Time Rows for a Specific Date**

This example extracts salary details for employees who worked on January 1, 2000.

Employment data from this employee table contains the ValidTime data column, JobDuration.

<table>
<thead>
<tr>
<th>EName</th>
<th>E_Id</th>
<th>Dept</th>
<th>Job_duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sania</td>
<td>1001</td>
<td>Dept1</td>
<td>1990-01-01, 2003-01-01</td>
</tr>
<tr>
<td>Sania</td>
<td>1001</td>
<td>Dept3</td>
<td>2003-01-01, UNTIL_CHANGED</td>
</tr>
<tr>
<td>Ash</td>
<td>1002</td>
<td>Dept1</td>
<td>1995-01-01, 2000-01-01</td>
</tr>
<tr>
<td>Ash</td>
<td>1002</td>
<td>Dept2</td>
<td>1999-01-01, 2010-01-01</td>
</tr>
</tbody>
</table>

Salary data is from the ValidTime column, SalaryPeriod.

<table>
<thead>
<tr>
<th>E_Id</th>
<th>Sal</th>
<th>SalaryPeriod</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>10000</td>
<td>1990-01-01, 2003-01-01</td>
</tr>
<tr>
<td>1001</td>
<td>20000</td>
<td>2003-01-01, 2010-01-01</td>
</tr>
<tr>
<td>1001</td>
<td>30000</td>
<td>2010-01-01, UNTIL_CHANGED</td>
</tr>
<tr>
<td>1002</td>
<td>25000</td>
<td>1995-01-01, 2010-01-01</td>
</tr>
</tbody>
</table>

Here is the query.

```
VALIDTIME AS OF DATE '2000-01-01'
SELECT E.EName as Name, S.Sal as Salary
FROM Employee E, Salary S
WHERE E.E_Id = S.E_Id;
```

It produces this data as the result.

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sania</td>
<td>10000</td>
</tr>
<tr>
<td>Ash</td>
<td>25000</td>
</tr>
</tbody>
</table>

**Example 3: Select Transaction-Time Rows for a Specific Date and Time**

This example extracts stock details as of a specific timestamp.

Data from this stock table contains a transaction-time dimension: the TransactionTime data column, RecordedTime.

<table>
<thead>
<tr>
<th>StockName</th>
<th>StockValue</th>
<th>RecordedTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teradata</td>
<td>38</td>
<td>2006-01-01 10:00:00.000000+00:00,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2006-01-01 12:10:00.000000+00:00</td>
</tr>
<tr>
<td>Teradata</td>
<td>37</td>
<td>2006-01-01 12:10:00.000000+00:00,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2006-01-03 10:00:00.000000+00:00</td>
</tr>
<tr>
<td>Teradata</td>
<td>40</td>
<td>2006-01-03 10:00:00.000000+00:00, UNTIL_CLOSED</td>
</tr>
</tbody>
</table>

Here is the query.

TRANSACTIONTIME AS OF TIMESTAMP '2006-01-02 12:10:10.000000+00:00'
SELECT * FROM Stock;
It produces this data as the result.

<table>
<thead>
<tr>
<th>StockName</th>
<th>StockValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teradata</td>
<td>37</td>
</tr>
</tbody>
</table>

**TENACITY= Data Set Option**

Specifies how many hours that FastExport, FastLoad, or MultiLoad continues to try to log on again to Teradata if the maximum number of Teradata utilities are already running.

- **Valid in:** DATA and PROC steps (when creating and appending to DBMS tables using SAS/ACCESS software)
- **Default:** 0 [FastLoad], 4 [FastExport, MultiLoad]
- **Data source:** Teradata
- **Tip:** The data set option has precedence over the LIBNAME option.
- **See:** DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option, DBSLICEPARM= system option, FASTEXPORT= LIBNAME option, Maximizing Teradata Load Performance, MULTILOAD= data set option, SLEEP= LIBNAME option, SLEEP= data set option, Using the TPT API

**Syntax**

```
TENACITY=number-of-hours
```

**Syntax Description**

`number-of-hours`

specifies the number of hours to continue to try again to log on to Teradata.

**Details**

Use this option to indicate to FastExport, FastLoad, or MultiLoad how long to continue retrying a logon operation when the maximum number of utilities are already running. (The maximum number of Teradata utilities that can run concurrently varies from 5 to 15, depending on the database server setting.) The default value for TENACITY= is four hours. The value specified for TENACITY= must be greater than zero.

Use TENACITY= with SLEEP=, which specifies the number of minutes that FastExport, FastLoad, or MultiLoad waits before it retries logging on to Teradata. SLEEP= and TENACITY= function very much like the SLEEP and TENACITY runtime options of the native Teradata FastExport, FastLoad, or MultiLoad utility.

Here is an example of the message that is written to the SAS log if the time period that TENACITY= specifies is exceeded.

```
ERROR:  MultiLoad failed unexpectedly with returncode 12
```

Check the FastExport, FastLoad, or MultiLoad log for more information about the cause of the FastExport, FastLoad, or MultiLoad failure. SAS does not receive any informational messages from Teradata in either of these situations:

- when the currently run FastExport, FastLoad, or MultiLoad process waits because the maximum number of utilities are already running
if FastExport, FastLoad, or MultiLoad is terminated because the time limit that TENACITY= specifies has been exceeded

The native Teradata FastExport, FastLoad, or MultiLoad utility sends messages associated with SLEEP= and TENACITY= only to the FastExport, FastLoad, or MultiLoad log. So nothing is written to the SAS log.

---

**TPT= Data Set Option**

Specifies whether SAS uses the TPT API to load data for Fastload, MultiLoad, or Multi-Statement insert requests.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software) |
| Default:  | YES |
| Data source: | Teradata |

*See:* BULKLOAD= LIBNAME option, BULKLOAD= data set option, LOGDB= LIBNAME option, "Maximizing Teradata Load and Read Performance", MULTILOAD= data set option, MULTISTMT= data set option, TPT_APPL_PHASE= data set option, TPT_BUFFER_SIZE= data set option, TPT_CHECKPOINT= data set option, TPT_DATA_ENCRYPTION= data set option, TPT_ERROR_TABLE_1= data set option, TPT_ERROR_TABLE_2= data set option, TPT_MAX_SESSIONS= LIBNAME option, TPT_MAX_SESSIONS= data set option, TPT_MIN_SESSIONS= data set option, TPT_PACKAGE= data set option, TPT_PACK_MAXIMUM= data set option, TPT_RESTART= data set option, TPT_TRACE_LEVEL= data set option, TPT_TRACE_LEVEL_INF= data set option, TPT_TRACE_OUTPUT= data set option, TPT_WORK_TABLE= data set option,

Maximizing Teradata Load Performance, “Using the TPT API”

---

**Syntax**

TPT=YES | NO

**Syntax Description**

**YES**

specifies that SAS uses the TPT API when Fastload, MultiLoad, or Multi-Statement insert is requested.

**NO**

specifies that SAS does not use the TPT API when Fastload, MultiLoad, or Multi-Statement insert is requested.

**Details**

By using the TPT API, you can load data into a Teradata table without working directly with such stand-alone Teradata utilities as Fastload, MultiLoad, or TPump. When TPT=NO, SAS uses the TPT API load driver for Fastload, the update driver for MultiLoad, and the stream driver for Multi-Statement insert.

When TPT=YES, sometimes SAS cannot use the TPT API due to an error or because it is not installed on the system. When this happens, SAS does not produce an error, but it still tries to load data using the requested load method (Fastload, MultiLoad, or Multi-Statement insert). To check whether SAS used the TPT API to load data, look for a similar message to this one in the SAS log:
NOTE: Teradata connection: TPT FastLoad/MultiLoad/MultiStatement insert has read n row(s).

Example: Load Data Using the TPT API

In this example, SAS data is loaded into Teradata using the TPT API. This is the default method of loading when Fastload, MultiLoad, or Multi-Statement insert is requested. SAS still tries to load data even if it cannot use the TPT API.

libname tera Teradata user=myusr1 pw=mypwd1;
/* Create data */
data testdata;
do i=1 to 100;
   output;
end;
run;
/* Load using FastLoad TPT. This note appears in the SAS log if SAS uses TPT.
NOTE: Teradata connection: TPT FastLoad has inserted 100 row{s}.
*/
data tera.testdata(FASTLOAD=YES TPT=YES);
set testdata;
run;

TPT_APPL_PHASE= Data Set Option

Specifies whether a load process that is being restarted has failed in the application phase.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>NO</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To use this option, you must first set TPT=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>Teradata</td>
</tr>
<tr>
<td>See:</td>
<td>Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD=LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, TPT=LIBNAME option, TPT= data set option, TPT_CHECKPOINT_DATA= data set option, TPT_RESTART= data set option</td>
</tr>
</tbody>
</table>

Syntax

TPT_APPL_PHASE=YES | NO

Syntax Description

YES
specifies that the Fastload or MultiLoad run that is being restarted has failed in the application phase. This is valid only when SAS uses the TPT API.

NO
specifies that the load process that is being restarted has not failed in the application phase.
Details

SAS can restart from checkpoints any Fastload, MultiLoad, and Multi-Statement insert that is run using the TPT API. The restart procedure varies: It depends on whether checkpoints were recorded and in which phase the step failed during the load process. Teradata loads data in two phases: the acquisition phase and the application phase. In the acquisition phase, data transfers from SAS to Teradata. After this phase, SAS has no more data to transfer to Teradata. If failure occurs after this phase, set TPT_APPL_PHASE=YES in the restart step to indicate that restart is in the application phase. (Multi-Statement insert does not have an application phase. Therefore, it does not need to be restarted if it fails after the acquisition phase.)

Use OBS=1 for the source data set when restart occurs in the application phase. When SAS encounters TPT_RESTART=YES and TPT_APPL_PHASE=YES, it initiates restart in the application phase. No data from the source data set is actually sent. If you use OBS=1 for the source data set, the SAS step completes as soon as it reads the first record. (It actually throws away the record because SAS already sent all data to Teradata for loading.)

Example: Restart after Failure

Here is a sample SAS program that failed after the acquisition phase.

```sas
libname x teradata user=mysrv1 pw=mypwd1;
data x.test(MULTILOAD=YES TPT=YES CHECKPOINT=7);
do i=1 to 20;
    output;
end;
run;
ERROR: Teradata connection: Failure occurred after the acquisition phase.
Restart outside of SAS using checkpoint data 14.

Set TPT_APPL_PHASE=YES to restart when failure occurs in the application phase because SAS has already sent all data to Teradata.

proc append base=x.test(MULTILOAD=YES TPT_RESTART=YES TPT_CHECKPOINT_DATA=14 TPT_APPL_PHASE=YES) data=test(obs=1);
run;
```

TPT_BUFFER_SIZE= Data Set Option

Specifies the output buffer size in kilobytes when SAS sends data to Teradata with Fastload or MultiLoad using the TPT API.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** 64
- **Requirement:** To use this option, you must first set TPT=YES.
- **Data source:** Teradata
- **See:** Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD= LIBNAME option, BULKLOAD= data set option, TPT= LIBNAME option, TPT= data set option
Syntax

TPT_BUFFER_SIZE=integer

Syntax Description

integer
specifies the size of data parcels in kilobytes from 1 through 64.

Details

You can use the output buffer size to control the amount of data that is transferred in each parcel from SAS to Teradata when using the TPT API. A larger buffer size can reduce processing overhead by including more data in each parcel. See your Teradata documentation for details.

TPT_CHECKPOINT_DATA= Data Set Option

Specifies the checkpoint data to return to Teradata when restarting a failed MultiLoad or Multi-Statement step that uses the TPT API.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirement: To use this option, you must first set TPT=YES and TPT_RESTART=YES.
Data source: Teradata
See: BULKLOAD= data set option, “Maximizing Teradata Load and Read Performance”, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT_APPL_PHASE= data set option, TPT_RESTART= data set option, “Using the TPT API”

Syntax

TPT_CHECKPOINT_DATA=checkpoint_data_in_error_message

Syntax Description

checkpoint_data_in_error_message
specifies the value to use to restart a failed MultiLoad or Multi-Statement step that uses the TPT API.

Details

SAS can restart from the last checkpoint any failed Fastload, MultiLoad, and Multi-Statement insert that are run using the TPT API. Teradata returns a checkpoint value each time MultiLoad or Multi-Statement records a checkpoint. The SAS log contains this value when a load fails. SAS must provide the same value as a data set option when it tries to restart the load process.

Here are the rules that govern restart.

- The TPT API does not return a checkpoint value when FastLoad records a checkpoint. Therefore, you do not need to set TPT_CHECKPOINT_VALUE= when you use FastLoad. Set TPT_RESTART= instead.
If the default error table name, work table name, or restart table name is overridden, SAS must use the same name while restarting the load process.

Teradata loads data in two phases: the acquisition phase and the application phase. In the acquisition phase, data transfers from SAS to Teradata. After this phase, SAS has no more data to transfer to Teradata. If failure occurs after this phase, set TPT_APPL_PHASE=YES while restarting. (Multi-Statement insert does not have an application phase. Therefore, it does not need to be restarted if it fails after the acquisition phase.) Use OBS=1 for the source data set because SAS has already sent the data to Teradata. Therefore, there is no need to send any more data.

If failure occurred before the acquisition phase ended and the load process recorded no checkpoints, you must restart the load process from the beginning by setting TPT_RESTART=YES. However, you do not need to set TPT_CHECKPOINT_VALUE= because no checkpoints were recorded. The error message in the SAS log provides all needed information for restart.

Examples

Example 1
In this example, assume that the MultiLoad step that uses the TPT API fails before the acquisition phase ends and no options were set to record checkpoints.

libname x teradata user=myusr1 pw=mypwd1;
data test;In
  do i=1 to 100;
    output;
  end;
run;
/* Set TPT=YES is optional because it is the default. */
data x.test(MULTILOAD=YES TPT=YES);
set test;
run;

This error message is sent to the SAS log. You do not need to set TPT_CHECKPOINT_DATA= because no checkpoints were recorded.

ERROR: Teradata connection: Correct error and restart as an APPEND process with option TPT_RESTART=YES. Since no checkpoints were taken, if the previous run used FIRSTOBS=n, use the same value in the restart.

Example 2
Here is an example of the restart step.

proc append data=test base=x.test(FASTLOAD=YES TPT=YES TPT_RESTART=YES);
run;

Example 3
In this next example, failure occurs after checkpoints are recorded.

libname tera teradata user=myusr1 pw=mypwd1;
/* Create data */
data testdata;
  do i=1 to 100;
    output;
  end;
runc
Here is the resulting error when it fails after loading 18 rows.

ERROR: Teradata connection: Correct error and restart as an APPEND process with option TPT_RESTART=YES. If the previous run used FIRSTOBS=n, use the value ( n-1+ 19 ) for FIRSTOBS in the restart. Otherwise use FIRSTOBS= 19. Also specify TPT_CHECKPOINT_DATA= 18.

You can restart the failed step with this code.

proc append base=x.test(MULTISTMT=YES TPT_RESTART=YES TPT_CHECKPOINT_DATA=18) data=test(firstobs=19);
run;

If failure occurs after the end of the acquisition phase, you must write a custom C++ program to restart from the point where it stopped.

Example 4
Here is a sample SAS program that failed after the acquisition phase and the resulting error message.

libname x teradata user=myusr1 pw=mypwd1;
data x.test(MULTILOAD=YES TPT=YES CHECKPOINT=7);
do i=1 to 20;
  output;
end;
run;
ERROR: Teradata connection: Failure occurred after the acquisition phase. Restart outside of SAS using checkpoint data 14.

Set TPT_APPL_PHASE=YES to restart when failure occurs in the application phase because SAS has already sent all data to Teradata.

proc append base=x.test(MULTILOAD=YES TPT_RESTART=YES TPT_CHECKPOINT_DATA=14 TPT_APPL_PHASE=YES) data=test(obs=1);
run;

TPT_DATA_ENCRYPTION= Data Set Option

Specifies whether to fully encrypt SQL requests, responses, and data when SAS sends data to Teradata for Fastload, MultiLoad, or Multi-Statement insert that uses the TPT API.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: NO
Requirement: To use this option, you must first set TPT=YES.
Data source: Teradata
See: Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option
Syntax

TPT_DATA_ENCRYPTION=YES | NO

Syntax Description

YES
specifies that all communication between the Teradata client and server is encrypted when using the TPT API.

NO
specifies that all communication between the Teradata client and server is not encrypted when using the TPT API.

Details

You can ensure that SQL requests, responses, and data that are transferred between the Teradata client and server are encrypted when using the TPT API. See your Teradata documentation for details.

TPT_ERROR_TABLE_1= Data Set Option

Specifies the name of the first error table for SAS to use when using the TPT API with Fastload or MultiLoad.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Default: table_name_ET
Restriction: This option is valid only when using the TPT API.
Requirement: To use this option, you must first set TPT=YES.
Data source: Teradata
See: Maximizing Teradata Load Performance, “Using the TPT API”, BULKLOAD=LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, TPT=LIBNAME option, TPT= data set option, TPT_ERROR_TABLE_2= data set option, TPT_LOG_TABLE= data set option, TPT_WORK_TABLE= data set option

Syntax

TPT_ERROR_TABLE_1=valid_teradata_table_name

Syntax Description

valid_teradata_table_name
specifies the name of the first error table for SAS to use when using the TPT API to load data with Fastload or MultiLoad.

Details

Fastload and MultiLoad require an error table to hold records that were rejected during the acquisition phase. If you do not specify an error table, Teradata appends "_ET" to the name of the target table to load and uses it as the first error table by default. You can override this name by setting TPT_ERROR_TABLE_1=. If you do this and the load step fails, you must specify the same name when restarting. For information about errors that are logged in this table, see your Teradata documentation.
The name that you specify in TPT_ERROR_TABLE_1= must be unique. It cannot be the name of an existing table unless it is in a restart scenario.

**Example: Specify Different Names for Two Error Tables**

In this example, a different name is provided for both the first and second error tables that Fastload and MultiLoad use with the TPT API.

```sas
libname tera teradata user=myusr1 pw=mypwd1;
/* Load using Fastload TPT. Use alternate names for the error tables. */
data tera.testdata(FASTLOAD=YES TPT_ERROR_TABLE_1=testerror1
  TPT_ERROR_TABLE_2=testerror2);
i=1;output; i=2;output;
run;
```

**TPT_ERROR_TABLE_2= Data Set Option**

Specifies the name of the second error table for SAS to use when using the TPT API with Fastload or MultiLoad.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>table_name_UV</td>
</tr>
<tr>
<td>Restriction:</td>
<td>This option is valid only when using the TPT API.</td>
</tr>
<tr>
<td>Requirement:</td>
<td>To use this option, you must first set TPT=YES.</td>
</tr>
<tr>
<td>Data source:</td>
<td>Teradata</td>
</tr>
<tr>
<td>See:</td>
<td>Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD=</td>
</tr>
<tr>
<td></td>
<td>LIBNAME option, BULKLOAD=data set option, MULTILOAD=data set option, TPT=</td>
</tr>
<tr>
<td></td>
<td>LIBNAME option, TPT=data set option, TPT_ERROR_TABLE_1=data set option,</td>
</tr>
<tr>
<td></td>
<td>TPT_LOG_TABLE=data set option, TPT_WORK_TABLE=data set option</td>
</tr>
</tbody>
</table>

**Syntax**

```
TPT_ERROR_TABLE_2=valid_teradata_table_name
```

**Syntax Description**

`valid_teradata_table_name`

specifies the name of the second error table for SAS to use when using the TPT API to load data with Fastload or MultiLoad.

**Details**

Fastload and MultiLoad require an error table to hold records that were rejected during the acquisition phase. If you do not specify an error table, Teradata appends "_UV" to the name of the target table to load and uses it as the second error table by default. You can override this name by setting TPT_ERROR_TABLE_2=. If you do this and the load step fails, you must specify the same name when restarting. For information about errors that are logged in this table, see your Teradata documentation.

The name that you specify in TPT_ERROR_TABLE_2= must be unique. It cannot be the name of an existing table unless it is in a restart scenario.
Example: Specify Different Names for First and Second Error Tables

In this example, a different name is provided for both the first and second error tables that Fastload and MultiLoad use with the TPT API.

libname tera teradata user=myusr1 pw=mypwd1;
/* Load using Fastload TPT. Use alternate names for the error tables. */
data tera.testdata(FASTLOAD=YES TPT_ERROR_TABLE_1=testerror1 TPT_ERROR_TABLE_2=testerror2);
i=1;output; i=2;output;
run;

TPT_LOG_TABLE= Data Set Option

Specifies the name of the restart log table for SAS to use when using the TPT API with Fastload, MultiLoad, or Multi-Statement insert.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** table_name_RS
- **Restriction:** This option is valid only when using the TPT API.
- **Requirement:** To use this option, you must first set TPT=YES.
- **Data source:** Teradata

**See:** Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option, TPT_ERROR_TABLE_1= data set option, TPT_ERROR_TABLE_2= data set option, TPT_WORK_TABLE= data set option

**Syntax**

```
TPT_LOG_TABLE=valid_teradata_table_name
```

**Syntax Description**

`valid_teradata_table_name`

specifies the name of the restart log table for SAS to use when using the TPT API to load data with Fastload or MultiLoad.

**Details**

To use this option, you must first set TPT=YES. This option is valid only when using the TPT API.

Fastload, MultiLoad, and Multi-Statement insert that use the TPT API require a restart log table. If you do not specify a restart log table, Teradata appends "_RS" to the name of the target load table and uses it as the restart log table by default. You can override this name by setting TPT_LOG_TABLE=. If you do this and the load step fails, you must specify the same name when restarting.

The name that you specify in TPT_LOG_TABLE= must be unique. It cannot be the name of an existing table unless it is in a restart scenario.
Example: Specify a Different Name for the Restart Log Table

In this example, a different name is provided for the restart log table that Multi-Statement uses with the TPT API.

```plaintext
libname tera teradata user=myusr1 pw=mypwd1;
/* Load using Fastload TPT. Use alternate names for the log table. */
data tera.testdata(MULTISTMT=YES TPT_LOG_TABLE=restarttab);
i=1;output; i=2;output;
run;
```

**TPT_MAX_SESSIONS= Data Set Option**

Specifies the maximum number of sessions for Teradata to use when using the TPT API with FastLoad, MultiLoad, or Multi-Statement insert.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** 4
- **Restriction:** This option is valid only when using the TPT API.
- **Requirement:** To use this option, you must first set TPT=YES.
- **Interaction:** [precedence] 1. TPT_MAX_SESSIONS= data set option, 2. TPT_MAX_SESSIONS= LIBNAME option, 3. SAS_TPT_MAX_SESSIONS environment variable, 4. the default value
- **Data source:** Teradata
- **Note:** The default value was changed to 4 for SAS 9.4.
- **See:** BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option, TPT_MAX_SESSIONS= LIBNAME option, TPT_MIN_SESSIONS= data set option, “Maximizing Teradata Load and Read Performance”, “Using the TPT API”

**Syntax**

```
TPT_MAX_SESSIONS=integer
```

**Syntax Description**

- **integer** specifies the maximum number of sessions for Teradata to use when using the TPT API to load data with FastLoad, MultiLoad, or Multi-Statement insert.

**Details**

You can control the number of sessions for Teradata to use when using the TPT API to load data with MultiLoad. The maximum value cannot be more than the number of available Access Module Processors (AMPs). See your Teradata documentation for details.
**TPT_MIN_SESSIONS= Data Set Option**

Specifies the minimum number of sessions for Teradata to use when using the TPT API with FastLoad, MultiLoad, or Multi-Statement insert.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** 1

**Restriction:** This option is valid only when using the TPT API.

**Requirement:** To use this option, you must first set TPT=YES.

**Data source:** Teradata

**See:** BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option, TPT_MAX_SESSIONS= LIBNAME option, TPT_MAX_SESSIONS= data set option, TPT_MIN_SESSIONS= data set option, “Maximizing Teradata Load and Read Performance”, “Using the TPT API”

---

**Syntax**

TPT_MIN_SESSIONS=integer

**Syntax Description**

*integer*

specifies the minimum number of sessions for Teradata to use when using the TPT API to load data with FastLoad, MultiLoad, or Multi-Statement insert.

**Details**

You can control the number of sessions that are required before using the TPT API to load data with MultiLoad. This value must be greater than zero and less than the maximum number of required sessions. See your Teradata documentation for details.

---

**TPT_PACK= Data Set Option**

Specifies the number of statements to pack into a Multi-Statement insert request when using the TPT API.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Default:** 20

**Restrictions:** The maximum value is 600. See your Teradata documentation for details.

This option is valid only when using the TPT API.

**Requirement:** To use this option, you must first set TPT=YES.

**Data source:** Teradata

**See:** Maximizing Teradata Load Performance, Using the TPT API, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option, TPT_PACK_MAXIMUM= data set option
Syntax

TPT_PACK=integer

**Syntax Description**

*integer*

specifies the number of statements to pack into a Multi-Statement insert request when using the TPT API.

---

**TPT_PACKMAXIMUM= Data Set Option**

Specifies whether to pack the maximum possible or default number of statements into Multi-Statement insert requests when using the TPT API.

| Valid in: | DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software) |
| Default: | NO |
| Restriction: | This option is valid only when using the TPT API. |
| Requirement: | To use this option, you must first set TPT=YES. |
| Data source: | Teradata |

See: Maximizing Teradata Load Performance, Using the TPT API, MULTISTMT= data set option, TPT=LIBNAME option, TPT= data set option, TPT_PACK= data set option, TPT_TRACE_LEVEL= data set option, TPT_TRACE_LEVEL_INF= data set option, TPT_TRACE_OUTPUT= data set option

---

**Syntax**

TPT_PACKMAXIMUM=YES | NO

**Syntax Description**

*YES*

specifies that the maximum possible number of statements be packed into Multi-Statement insert requests when using the TPT API.

*NO*

specifies that the default number of statements be packed into Multi-Statement insert requests when using the TPT API.

**Details**

When TPT_PACKMAXIMUM=YES, the maximum possible number of statements that can fit in a Multi-Statement request is determined dynamically. See your Teradata documentation for details.

---

**TPT_RESTART= Data Set Option**

Specifies that a failed FastLoad, MultiLoad, or Multi-Statement run that used the TPT API is being restarted.

| Valid in: | PROC steps (when accessing DBMS data using SAS/ACCESS software) |
Default: NO
Restriction: This option is valid only when using the TPT API.
Requirement: To use this option, you must first set TPT=YES.
Data source: Teradata
See: Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, BULKLOAD= data set option, TPT_APPL_PHASE= data set option, TPT_CHECKPOINT_DATA= data set option

Syntax

TPT_RESTART=YES | NO

Syntax Description

YES
specifies that the load process is being restarted.

NO
specifies that the load process is not being restarted.

Details

SAS can restart from checkpoints any FastLoad, MultiLoad, and Multi-Statement insert that are run using the TPT API. The restart procedure varies: It depends on whether checkpoints were recorded and in which phase the step failed during the load process. The error message in the log is extremely important and contains instructions on how to restart.

Here are the rules that govern restart.

- The TPT API does not return a checkpoint value when FastLoad records a checkpoint. Therefore, you do not need to set TPT_CHECKPOINT_VALUE= when you use FastLoad. Set TPT_RESTART= instead.

- If the default error table name, work table name, or restart table name is overridden, SAS must use the same name while restarting the load process.

- Teradata loads data in two phases: the acquisition phase and the application phase. In the acquisition phase, data transfers from SAS to Teradata. After this phase, SAS has no more data to transfer to Teradata. If failure occurs after this phase, set TPT_APPL_PHASE= YES while restarting. (Multi-Statement insert does not have an application phase. Therefore, it does not need to be restarted if it fails after the acquisition phase.) Use OBS=1 for the source data set because SAS has already sent the data to Teradata. Therefore, there is no need to send any more data.

- If failure occurred before the acquisition phase ended and the load process recorded no checkpoints, you must restart the load process from the beginning by setting TPT_RESTART=YES. However, you do not need to set TPT_CHECKPOINT_VALUE= because no checkpoints were recorded. The error message in the SAS log provides all needed information for restart.
Examples

Example 1
In this example, assume that the MultiLoad step that uses the TPT API fails before the acquisition phase ends and no options were set to record checkpoints.

```sas
libname x teradata user=myusr1 pw=mypwd1;
data test;
  do i=1 to 100;
    output;
  end;
run;
/* Set TPT=YES is optional because it is the default. */
data x.test(MULTILOAD=YES TPT=YES);
  set test;
run;
```

This error message is sent to the SAS log. You do not need to set TPT_CHECKPOINT_DATA= because no checkpoints were recorded.

ERROR: Teradata connection: Correct error and restart as an APPEND process with option TPT_RESTART=YES. Since no checkpoints were taken, if the previous run used FIRSTOBS=n, use the same value in the restart.

Example 2
Here is an example of the restart step.

```sas
proc append data=test base=x.test(MULTILOAD=YES TPT=YES TPT_RESTART=YES);
run;
```

Example 3
In this next example, failure occurs after checkpoints are recorded.

```sas
libname tera teradata user=myusr1 pw=mypwd1;
/* Create data */
data testdata;
  do i=1 to 100;
    output;
  end;
run;
/* Assume that this step fails after loading row 19. */
data x.test(MULTISTMT=YES CHECKPOINT=3);
  set testdata;
run;
```

Here is the resulting error when it fails after loading 18 rows.

ERROR: Teradata connection: Correct error and restart as
an APPEND process with option TPT_RESTART=YES. If the previous run used FIRSTOBS=n, use the value \( n-1+19 \) for FIRSTOBS in the restart. Otherwise use FIRSTOBS=19. Also specify TPT_CHECKPOINT_DATA= 18.

You can restart the failed step with this code.

```sas
proc append base=x.test(MULTISTMT=YES TPT_RESTART=YES TPT_CHECKPOINT_DATA=18) data=test(firstobs=19);
run;
```
If failure occurs after the end of the acquisition phase, you must write a custom C++ program to restart from the point where it stopped.

**Example 4**
Here is a sample SAS program that failed after the acquisition phase and the resulting error message.

```sas
libname x teradata user=myusr1 pw=mypwd1;
data x.test(MULTILOAD=YES TPT=YES CHECKPOINT=7);
do i=1 to 20;
output;
end;
rung;
ERROR: Teradata connection: Failure occurred after the acquisition phase. Restart outside of SAS using checkpoint data 14.

Set TPT_APPL_PHASE=YES to restart when failure occurs in the application phase because SAS has already sent all data to Teradata.

```sas
proc append base=x.test(MULTILOAD=YES TPT_RESTART=YES TPT_CHECKPOINT_DATA=14 TPT_APPL_PHASE=YES) data=test(obs=1);
rung;
```

You must always use TPT_CHECKPOINT_DATA= with TPT_RESTART= for MultLoad and Multi-Statement insert.

---

**TPT_TRACE_LEVEL= Data Set Option**

Specifies the required tracing level for sending data to Teradata and using the TPT API with Fastload, MultiLoad, or Multi-Statement insert.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option is valid only when using the TPT API.</td>
</tr>
<tr>
<td>Requirement</td>
<td>To use this option, you must first set TPT=YES.</td>
</tr>
<tr>
<td>Data source</td>
<td>Teradata</td>
</tr>
<tr>
<td>See</td>
<td>Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option, TPT_TRACE_LEVEL_INF= data set option, TPT_TRACE_OUTPUT= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

```sas
TPT_TRACE_LEVEL=integer
```

**Syntax Description**

`integer`

specifies the needed tracing level (1 to 9) when loading data to Teradata.
<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no tracing</td>
</tr>
<tr>
<td>2</td>
<td>operator-level general trace</td>
</tr>
<tr>
<td>3</td>
<td>operator-level command-line interface (CLI) trace</td>
</tr>
<tr>
<td>4</td>
<td>operator-level notify method trace</td>
</tr>
<tr>
<td>5</td>
<td>operator-level common library trace</td>
</tr>
<tr>
<td>6</td>
<td>all operator-level traces</td>
</tr>
<tr>
<td>7</td>
<td>Telnet API (TELAPI) layer general trace</td>
</tr>
<tr>
<td>8</td>
<td>PutRow or GetRow trace</td>
</tr>
<tr>
<td>9</td>
<td>operator log message information</td>
</tr>
</tbody>
</table>

**Details**

You can perform debugging by writing diagnostic messages to an external log file when loading data to Teradata using the TPT API. If you do not specify a name in TPT_TRACE_OUTPUT= for the log file, a default name is generated using the current timestamp. See your Teradata documentation for details.

---

**TPT_TRACE_LEVEL_INF= Data Set Option**

Specifies the tracing level for the required infrastructure for sending data to Teradata and using the TPT API with Fastload, MultiLoad, or Multi-Statement insert.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** 1
- **Restriction:** This option is valid only when using the TPT API.
- **Requirement:** To use this option, you must first set TPT=YES.
- **Data source:** Teradata
- **See:** Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option, TPT_TRACE_LEVEL= data set option, TPT_TRACE_OUTPUT= data set option

**Syntax**

\[ TPT\_TRACE\_LEVEL\_INF=integer \]
Syntax Description

integer

does not affect the data loading performance.

<table>
<thead>
<tr>
<th>Tracing Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>no tracing</td>
</tr>
<tr>
<td>11</td>
<td>operator-level general trace</td>
</tr>
<tr>
<td>12</td>
<td>operator-level command-line interface (CLI) trace</td>
</tr>
<tr>
<td>13</td>
<td>operator-level notify method trace</td>
</tr>
<tr>
<td>14</td>
<td>operator-level common library trace</td>
</tr>
<tr>
<td>15</td>
<td>all operator-level traces</td>
</tr>
<tr>
<td>16</td>
<td>Telnet API (TELAPI) layer general trace</td>
</tr>
<tr>
<td>17</td>
<td>PutRow or GetRow trace</td>
</tr>
<tr>
<td>18</td>
<td>operator log message information</td>
</tr>
</tbody>
</table>

Details

You can perform debugging by writing diagnostic messages to an external log file when loading data to Teradata using the TPT API. If you do not specify a name in PT_TRACE_OUTPUT= for the log file, a default name is generated using the current timestamp. See your Teradata documentation for details.

TPT_TRACE_OUTPUT= Data Set Option

Specifies the name of the external file for SAS to use for tracing when using the TPT API with Fastload, MultiLoad, or Multi-Statement insert.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: driver_name_timestamp

Restriction: This option is valid only when using the TPT API.

Requirement: To use this option, you must first set TPT=YES.

Data source: Teradata

See: Maximizing Teradata Load Performance, Using the TPT API, BULKLOAD= LIBNAME option, BULKLOAD= data set option, MULTILOAD= data set option, MULTISTMT= data set option, TPT= LIBNAME option, TPT= data set option, TPT_TRACE_LEVEL= data set option, TPT_TRACE_LEVEL_INF= data set option, TPT_PACKMAXIMUM= data set option
Syntax

TPT_TRACE_OUTPUT=filename

Syntax Description

filename

specifies the name of the external file to use for tracing. The name must be a valid filename for the operating system.

Details

You can save diagnostic messages to an external log file when loading data to Teradata using the TPT API. If you request tracing but specify no name in TPT_TRACE_OUTPUT= for the log file, a default name is generated using the name of the driver and the current timestamp. Otherwise, the name that you specify is used for tracing messages. If the file already exists, it is overwritten. See your Teradata documentation for details.

TPT_WORK_TABLE= Data Set Option

Specifies the name of the work table for SAS to use when using the TPT API with MultiLoad.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Default: table_name_WT

Restriction: This option is valid only when using the TPT API.

Requirement: To use this option, you must first set TPT=YES.

Data source: Teradata

See: Maximizing Teradata Load Performance, Using the TPT API, MULTILOAD= data set option, TPT= LIBNAME option, TPT= data set option, TPT_ERROR_TABLE_1= data set option, TPT_ERROR_TABLE_2= data set option, TPT_LOG_TABLE= data set option

Syntax

TPT_WORK_TABLE=valid_teradata_table_name

Syntax Description

valid_teradata_table_name

specifies the name of the work table for SAS to use when using the TPT API to load data with MultiLoad.

Details

MultiLoad inserts that use the TPT API require a work table. If you do not specify a work table, Teradata appends "_WT" to the name of the target table to load and uses it as the work table by default. You can override this name by setting TPT_WORK_TABLE=. If you do this and the load step fails, you must specify the same name when restarting.

The name that you specify in TPT_WORK_TABLE= must be unique. It cannot be the name of an existing table unless it is in a restart scenario.
Example: Specify a Different Name for the Work Table

In this example, a different name is provided for the work table that MultiLoad uses with the TPT API.

```sas
libname tera teradata user=myusr1 pw=mypwd1;
/* Load using Multiload TPT. Use alternate names for the work table. */
data tera.testdata(MULTILOAD=YES TPT_WORK_TABLE=worktab);
i=1;output; i=2;output;
run;
```

TRANSCODE_FAIL= Data Set Option

Lets you specify how to handle processing and notification of transcoding errors.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** LIBNAME option setting
- **Restriction:** This option is not available for use with SAS Embedded Process.
- **Data source:** Hadoop
- **Tip:** You can use TRANSCODE_FAIL= to determine whether you want to halt or continue processing when transcoding errors are encountered.
- **See:** TRANSCODE_FAIL= LIBNAME option

**Syntax**

```
TRANSCODE_FAIL=<ERROR> | <WARNING> | <SILENT>
```

**Optional Arguments**

- **ERROR**
  - stops processing data and provides an informative error message.

- **WARNING**
  - continues processing of data but provides an informative error message.

- **SILENT**
  - continues processing of data but suppresses messages.

TRAP151= Data Set Option

Enables removal of columns that cannot be updated from a FOR UPDATE OF clause so that update of columns can proceed as normal.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Data source:** DB2 under z/OS
Syntax

**TRAP151= YES | NO**

**Syntax Description**

**YES**

removes the non-updatable column that is designated in the error-151 and reprepares the statement for processing. This process is repeated until all columns that cannot be updated are removed, and all remaining columns can be updated.

**NO**

disables TRAP151=. TRAP151= is disabled by default. It is not necessary to specify NO.

**Examples**

**Example 1: SAS Log for TRAP151=YES**

In this example, the SASTRACE=',,,d' option is used so that you can see what occurs when TRAP151=YES:

```sas
proc fsedit data=x.v4(trap151=yes);
run;
SELECT * FROM V4 FOR FETCH ONLY
SELECT * FROM V4 FOR FETCH ONLY
SELECT "A","X","Y","B","Z","C" FROM V4 FOR UPDATE OF "A","X","Y","B","Z","C"
DB2 SQL Error, sqlca->sqlcode=-151
WARNING: SQLCODE -151: repreparing SELECT as:
  SELECT "A","X","Y","B","Z","C" FROM V4 FOR UPDATE OF "A","Y","B","Z","C"
DB2 SQL Error, sqlca->sqlcode=-151
WARNING: SQLCODE -151: repreparing SELECT as:
  SELECT "A","X","Y","B","Z","C" FROM V4 FOR UPDATE OF "A","B","Z","C"
DB2 SQL Error, sqlca->sqlcode=-151
WARNING: SQLCODE -151: repreparing SELECT as:
  SELECT "A","X","Y","B","Z","C" FROM V4 FOR UPDATE OF "A","B","C"
COMMIT WORK
NOTE: The PROCEDURE FSEDIT used 0.13 CPU seconds and 14367K.
```

**Example 2: SAS Log for TRAP151=NO**

The next example features the same code with TRAP151 turned off:

```sas
proc fsedit data=x.v4(trap151=no);
run;
SELECT * FROM V4 FOR FETCH ONLY
SELECT * FROM V4 FOR FETCH ONLY
SELECT "A","X","Y","B","Z","C" FROM V4 FOR UPDATE OF "A","X","Y","B","Z","C"
DB2 SQL Error, sqlca->sqlcode=-151
ERROR: DB2 prepare error; DSNT4081 SQLCODE= -151, ERROR;
THE UPDATE STATEMENT IS INVALID BECAUSE THE CATALOG DESCRIPTION OF COLUMN C INDICATES THAT IT CANNOT BE UPDATED.
COMMIT WORK
NOTE: The SAS System stopped processing this step because of errors.
NOTE: The PROCEDURE FSEDIT used 0.08 CPU seconds and 14367K.
```
### UPDATE_ISOLATION_LEVEL= Data Set Option

Defines the degree of isolation of the current application process from other concurrently running application processes.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** LIBNAME option setting
- **Data source:** DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, MySQL, ODBC, OLE DB, Oracle, PostgreSQL, Sybase, Teradata, Vertica
- **See:** READ_ISOLATION_LEVEL= LIBNAME option, READ_LOCK_TYPE= LIBNAME option, READ_LOCK_TYPE= data set option, UPDATE_ISOLATION_LEVEL= LIBNAME option, UPDATE_LOCK_TYPE= LIBNAME option, UPDATE_LOCK_TYPE= data set option, and DBMS-specific locking information in the reference section for your SAS/ACCESS interface

### Syntax

UPDATE_ISOLATION_LEVEL= DBMS-specific-value

### Details

The degree of isolation defines the degree to which these items are affected:

- Rows that the current application reads and updates are available to other concurrently executing applications.
- Update activity of other concurrently executing application processes can affect the current application.

### UPDATE_LOCK_TYPE= Data Set Option

Specifies how data in a DBMS table is locked during an update transaction.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** LIBNAME option setting
- **Data source:** DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, OLE DB, Oracle, PostgreSQL, Sybase, Sybase IQ, Teradata, Vertica
- **Tip:** If you omit UPDATE_LOCK_TYPE=, you receive either the default action for the DBMS that you are using, or a lock for the DBMS that was set with the LIBNAME statement. You can set a lock for one DBMS table by using the data set option or for a group of DBMS tables by using the LIBNAME option.
- **See:** READ_ISOLATION_LEVEL= LIBNAME option, READ_ISOLATION_LEVEL= data set option, READ_LOCK_TYPE= LIBNAME option, READ_LOCK_TYPE= data set option, UPDATE_ISOLATION_LEVEL= LIBNAME option, UPDATE_ISOLATION_LEVEL= data set option, UPDATE_LOCK_TYPE= LIBNAME option, and DBMS-specific locking information in the reference section for your SAS/ACCESS interface
Syntax

**UPDATE_LOCK_TYPE**=ROW | PAGE | TABLE | NOLOCK | VIEW

**Syntax Description**

ROW [valid for DB2 under UNIX and PC Hosts, Microsoft SQL Server, ODBC, OLE DB, Oracle, PostgreSQL, Vertica]
locks a row if any of its columns are going to be updated.

PAGE [valid for Sybase]
locks a page of data, which is a DBMS-specific number of bytes.

TABLE
locks the entire DBMS table. (This value is valid in the DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, Oracle, and Teradata interfaces.)

NOLOCK
does not lock the DBMS table, page, or any rows when reading them for update. (This value is valid in the Microsoft SQL Server, ODBC, Oracle, and Sybase interfaces.)

VIEW
locks the entire DBMS view. (This value is valid in the Teradata interface.)

---

**UPDATE_MODE_WAIT= Data Set Option**

Specifies during SAS/ACCESS Update operations whether the DBMS waits to acquire a lock or fails your request when a different user has locked the DBMS resource.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Default:** LIBNAME option setting
- **Data source:** Teradata
- **See:** UPDATE_MODE_WAIT= LIBNAME option. Locking in the Teradata Interface

**Syntax**

**UPDATE_MODE_WAIT**=YES | NO

**Syntax Description**

YES
specifies that Teradata waits to acquire the lock, so SAS/ACCESS waits indefinitely until it can acquire the lock.

NO
specifies that Teradata fails the lock request if the specified DBMS resource is locked.

**Details**

If you specify UPDATE_MODE_WAIT=NO and if a different user holds a restrictive lock, then your SAS step fails and SAS/ACCESS continues the job by processing the
next step. If you specify UPDATE_MODE_WAIT=YES, SAS/ACCESS waits indefinitely until it can acquire the lock.

A restrictive lock means that a different user is holding a lock that prevents you from obtaining your desired lock. Until the other user releases the restrictive lock, you cannot obtain your lock. For example, another user's table-level WRITE lock prevents you from obtaining a READ lock on the table.

Use SAS/ACCESS locking options only when Teradata standard locking is undesirable.
For more information, see the locking topic in the Teradata section.

---

**UPDATE_SQL= Data Set Option**

Determines which method to use to update and delete rows in a data source.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>Data source:</td>
<td>Microsoft SQL Server, ODBC, PostgreSQL, Vertica</td>
</tr>
</tbody>
</table>

**Syntax**

```
UPDATE_SQL=YES | NO
```

**Syntax Description**

YES

specifies that SAS/ACCESS uses Current-of-Cursor SQL to update or delete rows in a table.

NO

specifies that SAS/ACCESS uses the SQLSetPos() API to update or delete rows in a table.

**Details**

This is the update and delete equivalent of the INSERT_SQL= data set option.

---

**UPDATEBUFF= Data Set Option**

Specifies the number of rows that are processed in a single DBMS Update or Delete operation.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>Data source:</td>
<td>Oracle</td>
</tr>
</tbody>
</table>

**Syntax**

```
UPDATEBUFF=positive-integer
```
Syntax Description

**positive-integer**

is the maximum value that is allowed by the DBMS.

Details

When updating with the VIEWTABLE window or PROC FSVIEW, use UPDATEBUFF=1 to prevent the DBMS interface from trying to update multiple rows. By default, these features use record-level locking and update only one observation at a time. They lock only the observation that is currently being edited.

---

**UPSERT= Data Set Option**

Specifies whether a Teradata MultiLoad upsert should take place.

- **Valid in:** PROC APPEND (when accessing DBMS data using SAS/ACCESS software)
- **Default:** NO
- **Requirement:** MULTILOAD=YES must be specified.
- **Data source:** Teradata
- **See:** MULTILOAD= data set option, , UPSERT_CONDITION= data set option, UPSERT_WHERE= data set option

Syntax

UPSERT=YES | NO

Syntax Description

YES

uses the Teradata MultiLoad upsert feature.

NO

performs inserts without upserts.

Details

The MultiLoad bulk-load facility supports the Teradata upsert feature. When an update fails because a target row does not exist, the upsert feature updates the table by inserting the missing row.

The upsert feature performs a combination of updates and inserts in one step. When updating a master table with a transaction table, an UPDATE statement is first issued on the master table using a row of data from the transaction table. If no target row exists to satisfy the update, an INSERT statement adds the transaction row to the master table.

Upsert requires a WHERE clause on the primary index of the target table to identify the target rows to be updated. When UPSERT=YES, Teradata builds a WHERE condition, by default, based on the primary index columns of the target table.

To add conditions to the WHERE condition, use UPSERT_CONDITION=. 
Example

```sas
libname x teradata user=user pw=pw;
/* Create a master table-Column where i is the primary index */
data x.master;
i=1;
j=1;
output;
i=2;
j=2;
output;
run;

/* Create a transaction table */
data x.transaction;
i=1;
j=99;
output;
run;

/* Running Upsert operates an Update statement based on the primary index column i. All other columns in the master are updated with transaction data. A snippet of the generated MultiLoad script is included: */
.DML Label SASDML DO INSERT FOR MISSING UPDATE ROWS;
UPDATE "master"
SET "j"=:"j";
WHERE "i"=:"i";
INSERT "master"("i","j") VALUES (:"i",:"j");
*/

proc append base=x.master (MULTILOAD=YES UPSERT=YES) data=x.transaction;
run;
```

**UPSERT_CONDITION= Data Set Option**

Specifies one or more conditions to add to a WHERE condition for a Teradata MultiLoad upsert.

- **Valid in:** PROC APPEND only when appending Teradata tables (when using SAS/ACCESS software).
- **Default:** none
- **Requirement:** MULTILOAD=YES must be specified.
- **Data source:** Teradata
- **See:** MULTILOAD= data set option, UPSERT= data set option, UPSERT_WHERE= data set option

**Syntax**

```sas
UPSERT_CONDITION=condition(s)-to-append
```
Syntax Description

(condition(s)-to-append)

specifies one or more conditions that are appended to UPSERT_WHERE= for upsert processing.

Details

By default, MultiLoad upsert uses only the primary index columns of the target table to generate the WHERE condition for upsert processing.

When you need to add conditions to the generated WHERE condition, specify them using UPSERT_CONDITION=. Add the AND keyword between the generated WHERE condition and the specified UPSERT_CONDITION= data set option.

Example

```sas
libname x teradata user=user pw=pw;
/
Create a master table-Column i where i is the primary index */
data x.master;
i=1;
j=1;
k=1;
output;
i=1;
j=2;
k=2;
output;
run;
/
Create a transaction table */
data x.transaction;
i=1;
j=99;output;
run;
/
Running Upser with UPSERT_CONDITION= "j=2" causes "AND j=2" tp be appended to the Upser Where clause. A snippet of the generated MultiLoad script is included:

.DML Label SASDML DO INSERT FOR MISSING UPDATE ROWS;
UPDATE "master"
   SET "j"=:"j","k"="k";
WHERE "i"=:"i" AND j=2;
INSERT "master"("i","j","k") VALUES (:"i",:"j",:"k");
*/
```

```sas
proc append base=x.master (MULTILOAD=YES UPsert=YES UPSERT_CONDITION="j=2")
data=x.transaction;
run;
```

UPSERT_WHERE= Data Set Option

Specifies which columns in the master table are used to generate a WHERE condition for a Teradata MultiLoad upsert.
Valid in: PROC APPEND (when accessing DBMS data using SAS/ACCESS software)
Default: none
Requirement: MULTILOAD=YES must be specified.
Data source: Teradata
See: MULTILOAD= data set option, UPSERT= data set option, UPSERT_CONDITION= data set option

Syntax

**UPSERT WHERE=list-of-column(s)**

**Syntax Description**

*list-of-column(s)* specifies one or more columns that are used to generate conditions for upsert processing of Teradata tables.

**Details**

By default, MultiLoad upsert uses only the primary index columns of the target table to generate the WHERE condition for upsert processing.

Upsert processing requires a WHERE clause on the primary index of the target table to identify the target rows to be updated. When you are using the upsert feature, a WHERE condition is built, by default, based on the primary index columns of the target table.

When you need additional columns to identify target rows, use **UPSERT WHERE=** to list the columns to be used. Note that you need to include the columns that make up the primary index of the target table.

**Example**

```plaintext
libname x teradata user=user pw=pw;
/* Create a master table - Column i where i is the primary index */
data x.master;
i=1;
j=1;
k=1;
output;
i=1;
j=2;
k=2;
output;
run;

/* Create a transaction table */
data x.transaction;
i=1;
j=2;
k=99;
output;
run;

/* Running Upsert with UPSERT WHERE=(i j) generates an update Where clause
```
based on columns i and j. All other columns in the master are updated with transaction data.

A snippet of the generated MultiLoad script is included:

```sas
.DML Label SASDML DO INSERT FOR MISSING UPDATE ROWS;
UPDATE "master"
    SET "k"=:"k";
WHERE "i"=:"i" AND "j"=:"j";
INSERT "master"("i","j","k") VALUES ("i","j","k");
/*
proc append base=x.master (MULTILOAD=YES  UPSERT=YES   UPSERT_WHERE=(i j))
    data=x.transaction;
run;
```
Chapter 13
Macro Variables and System Options for Relational Databases

Introduction to Macro Variables and System Options

This section describes macro variables on page 445 and system options on page 447 that you can use with SAS/ACCESS software. It describes only those components of the macro facility that depend on SAS/ACCESS engines. Most features of the SAS macro facility are portable.

For more information about the macro facility, see SAS Macro Language: Reference. For more information about SAS system options, see SAS System Options: Reference.

Macro Variables for Relational Databases

SYSDBMSG, SYSDBRC, SQLXMSG, and SQLXRC are automatic SAS macro variables. The SAS/ACCESS engine and your DBMS determine their values. Initially, SYSDBMSG and SQLXMSG are blank, and SYSDBRC and SQLXRC are set to 0.

SAS/ACCESS generates several return codes and error messages while it processes your programs. This information is available to you through these SAS macro variables.
SYSDBMSG
contains DBMS-specific error messages that are generated when you use SAS/ACCESS software to access your DBMS data.

SYSDBRC
contains DBMS-specific error codes that are generated when you use SAS/ACCESS software to access your DBMS data. Error codes that are returned are text, not numbers.

You can use these variables anywhere while you are accessing DBMS data. Only one set of macro variables is provided, however. So it is possible that, if tables from two different DBMSs are accessed, it might not be clear from which DBMS the error message originated. To address this problem, the name of the DBMS is inserted at the beginning of the SYSDBMSG macro variable message or value. The contents of the SYSDBMSG and SYSDBRC macro variables can be printed in the SAS log by using the %PUT macro. They are reset after each SAS/ACCESS LIBNAME statement, DATA step, or procedure is executed. In the statement below, %SUPERQ masks special characters such as &, %, and any unbalanced parentheses or quotation marks that might exist in the text stored in the SYSDBMSG macro.

%put %superq(SYSDBMSG)

These special characters can cause unpredictable results if you use this statement:

%put &SYSDBMSG

It is more advantageous to use %SUPERQ.

If you try to connect to Oracle and use the incorrect password, you receive the messages shown in this output.

Log 13.1 SAS Log for an Oracle Error

2? libname mydblib oracle user=pierre pass=paris path="mypath";
ERROR: Oracle error trying to establish connection. Oracle error is ORA-01017: invalid username/password; logon denied
ERROR: Error in the LIBNAME or FILENAME statement.
3? %put %superq(sysdbmsg);
Oracle: ORA-01017: invalid username/password; logon denied
4? %put &sysdbrc;
-1017
5?

You can also use SYMGET to retrieve error messages:

msg=symget("SYSDBMSG");

Here is an example.

data_null_; msg=symget("SYSDBMSG"); put msg; run;

The SQL pass-through facility generates return codes and error messages that are available to you through these SAS macro variables:

SQLXMSG
contains DBMS-specific error messages.
SQLXRC contains DBMS-specific error codes.

You can use SQLXMSG and SQLXRC only through explicit pass-through with the SQL pass-through facility. See Return Codes on page 472.

You can print the contents of SQLXMSG and SQLXRC in the SAS log by using the %PUT macro. SQLXMSG is reset to a blank string, and SQLXRC is reset to 0 when any SQL pass-through facility statement is executed.

---

**System Options for Relational Databases**

**Available System Options**

Here are the available systems options.

No SAS/ACCESS interface support is available for the REPLACE= system option. (See SAS System Options: Reference.)

<table>
<thead>
<tr>
<th>SAS System Options</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2CATALOG=</td>
<td>SYSIBM</td>
</tr>
<tr>
<td>DBFMTIGNORE</td>
<td>NODBFMTIGNORE</td>
</tr>
<tr>
<td>DBIDIRECTEXEC</td>
<td>NODBDIRECTEXEC</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 [DB2 under z/OS, Oracle, and Teradata]</td>
</tr>
<tr>
<td></td>
<td>THREADED_APPS,2 or THREADED_APP,3 [DB2 under UNIX and PC Hosts, Informix, Microsoft SQL Server, ODBC, and Sybase, Sybase IQ]</td>
</tr>
<tr>
<td>DBSRVTP=</td>
<td>NONE</td>
</tr>
<tr>
<td>REPLACE=</td>
<td>No SAS/ACCESS interface support is available. See SAS System Options: Reference.</td>
</tr>
<tr>
<td>SASTRACE=</td>
<td>none</td>
</tr>
<tr>
<td>SASTRACELOC=</td>
<td>stdout</td>
</tr>
<tr>
<td>SQLGENERATION=</td>
<td>Specifies whether and when SAS procedures generate SQL for in-database processing of source data.</td>
</tr>
<tr>
<td>SQLMAPPUTTO=</td>
<td>Specifies whether the PUT function is mapped to the SAS_PUT() function for a database, possible also where the SAS_PUT() function is mapped.</td>
</tr>
<tr>
<td>VALIDVARNAME=</td>
<td>Controls the type of SAS variable names that can be used or created during a SAS session.</td>
</tr>
</tbody>
</table>
Dictionary

**DB2CATALOG= SAS System Option**

Overrides the default owner of DB2 catalog tables.

**Valid in:** OPTIONS statement  
**Category:** Databases: DB2  
**Default:** SYSIBM  
**Restriction:** This option applies to only the local DB2 subsystem. So when you set the LOCATION= or SERVER= connection option in the LIBNAME statement, the SAS/ACCESS engine always uses SYSIBM as the default value.

**Data source:** DB2 under z/OS  
**See:** LOCATION= connection option, SERVER= connection option

### Syntax

**Syntax**

```
DB2CATALOG=SYSIBM | catalog-owner
```

### Syntax Description

- **SYSIBM**  
  specifies the default catalog owner.

- **catalog-owner**  
  specifies a different catalog owner from the default.

### Details

The default value for this option is initialized when SAS is installed. You can override the default only when these conditions are met:

- SYSIBM cannot be the owner of the catalog that you want to access.
- Your site must have a shadow catalog of tables (one to which all users have access).
- You must set DB2CATALOG= in the restricted options table and then rebuild the table.

---

**DBFMTIGNORE SAS System Option**

Specifies whether to ignore numeric formats.

**Valid in:** configuration file, SAS invocation, OPTIONS statement, SAS System Options window  
**Categories:** Files: SAS Files  
**Input Control:** Data Processing
**DBFMTIGNORE | NODBFMTIGNORE**

**Syntax Description**

**DBFMTIGNORE**

- Specifies that numeric formats are ignored and FLOAT data type is created.

**NODBFMTIGNORE**

- Specifies that numeric formats are used.

**Details**

You normally use numeric formats to specify a database data type when processing output. SAS takes all nonnumeric formats (such as date, time, datetime, and char) as hints when it processes output. So use this option to ignore numeric formats and create a FLOAT data type instead. For example, the SAS/ACCESS engine creates a table with a column type of INT for a SAS variable with a format of 5.0.

---

**DBIDIRECTEXEC SAS System Option**

Lets the SQL pass-through facility optimize handling of SQL statements by passing them directly to the database for execution.

- **Valid in:** configuration file, SAS invocation, OPTIONS statement, SAS System Options window
- **Categories:** Files: External Files, System Administration: Performance
- **Default:** NODBIDIRECTEXEC
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, PostgreSQL, SAP HANA, Sybase, Sybase IQ, Teradata, Vertica
- **See:** DIRECT_EXE= LIBNAME option

**Syntax**

**DBIDIRECTEXEC | NODBIDIRECTEXEC**

**Syntax Description**

**DBIDIRECTEXEC**

- Indicates that the SQL pass-through facility optimizes handling of SQL statements by passing them directly to the database for execution, which optimizes performance.
Using this option, you can process CREATE TABLE AS SELECT and DELETE statements.

**NODBIDIRECTEXEC**
indicates that the SQL pass-through facility does not optimize handling of SQL statements.

**Details**
This option applies to all hosts and all SAS/ACCESS engines. You can use it to significantly improve CPU, input, and output performance.

Certain database-specific criteria exist for passing SQL statements to the DBMS. These criteria are the same as the criteria that exist for passing joins. For details for your DBMS, see “Passing Joins to the DBMS” on page 45 and “When Passing Joins to the DBMS Will Fail” on page 47.

When these criteria are met, a database can process the `CREATE TABLE table-name AS SELECT` statement in a single step instead of as three separate statements (CREATE, SELECT, and INSERT). For example, if multiple librefs point to different data sources, the statement is processed normally, regardless of how you set this option. However, when you enable it, PROC SQL sends the CREATE TABLE AS SELECT statement to the database.

You can also send a DELETE statement directly to the database for execution, which can improve CPU, input, and output performance.

Once a system administrator sets the default for this option globally, users can override it within their own configuration file.

When you specify `DBIDIRECTEXEC=`, PROC SQL can pass this statement directly to the database:

```
CREATE TABLE table-name AS SELECT query
```

Before an SQL statement can be processed, all librefs that are associated with the statement must reference compatible data sources. For example, a CREATE TABLE AS SELECT statement that creates an Oracle table by selecting from a SAS table is not sent to the database for execution. The reason is that the data sources are not compatible. The libref must also use the same database server for all compatible data sources.

**Examples**

**Example 1: Create a Temporary Table**
This example creates a temporary table from a SELECT statement using the DBIDIRECTEXEC system option.

```
libname lib1 db2 user=myusr1 password=mypwd1 datasrc=sample connection=global;
libname lib2 db2 user=myusr2 password=mypwd2 datasrc=sample
   connection=global dbmstemp=yes;
data lib1.tab1;
a=1;
b='one';
run;
options dbidirectexec sastraceloc=saslog;
proc sql;
   create table lib2.tab1 as
   select * from lib1.tab1;
quit;
```
Example 2: Reference One Database, Use Different Schemas
In this example, two librefs point to the same database server but use different schemas.

```
libname lib1 db2 user=myusr1 password=mypwd1 datasrc=sample;
libname lib2 db2 user=myusr2 password=mypwd2 datasrc=sample;
data lib1.tab1;
a=1;
b='one';
rn;
options dbidirectexec sastraceloc=saslog;
proc sql;
  create table lib2.tab2 as
  select * from lib1.t1;
quit;
```

Example 3: Pass a Statement Directly to the Database
This example shows how a statement can be passed directly to the database for execution, if you specify DBIDIRECTEXEC.

```
libname company oracle user=myusr1 pw=mypwd1 path=mydb;
proc sql;
  create table company.hr_tab as
  select * from company.emp
  where deptid = 'HR';
quit;
```

DBSLICEPARM= SAS System Option
Controls the scope of DBMS threaded Reads and the number of threads.

- **Valid in:** configuration file, SAS invocation, OPTIONS statement, SAS System Options window
- **Category:** System Administration: Performance
- **Defaults:** THREADED_APPS, none [Greenplum, HAWQ] THREADED_APPS,2 [DB2 under z/OS, Oracle, Teradata] THREADED_APPS,2or 3 [DB2 under UNIX and PC Hosts, Informix, Microsoft SQL Server, ODBC, Sybase, Sybase IQ]
- **Data source:** DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, HAWQ, Informix, Microsoft SQL Server, ODBC, Oracle, Sybase, Sybase IQ, Teradata
- **Notes:** Support for Greenplum was added in the second maintenance release for SAS 9.4. Support for HAWQ was added in the third maintenance release of SAS 9.4.
- **See:** DBSLICE= data set option, DBSLICEPARM= LIBNAME option, DBSLICEPARM= data set option, SLEEP= LIBNAME option, SLEEP= data set option, TENACITY= LIBNAME option, TENACITY= data set option

**Syntax**

```
DBSLICEPARM=NONE | THREADED_APPS | ALL
DBSLICEPARM=(NONE | THREADED_APPS | ALL <max-threads>)
DBSLICEPARM=(NONE | THREADED_APPS | ALL<,max-threads>)
```
**Syntax Description**

**NONE**
- disables DBMS threaded Read. SAS reads tables on a single DBMS connection, as it did with SAS 8 and earlier.

**THREADED_APPS**
- makes fully threaded SAS procedures (threaded applications) eligible for threaded Reads.

**ALL**
- makes all read-only librefs eligible for threaded Reads. This includes SAS threaded applications, as well as the SAS DATA step and numerous SAS procedures.

**max-threads**
- a positive integer value that specifies the maximum number of connections per table read. The second parameter of the option determines the number of threads to read the table in parallel. The number of partitions on the table determine the number of connections made to the Oracle server for retrieving rows from the table. A partition or portion of the data is read on each connection. The combined rows across all partitions are the same regardless of the number of connections. That is, changes to the number of connections do not change the result set. Increasing the number of connections instead redistributes the same result set across more connections.

There are diminishing returns when increasing the number of connections. With each additional connection, more burden is placed on the DBMS, and a smaller percentage of time saved on the SAS step. See the DBMS-specific reference section for details about partitioned reads before using this parameter.

**Details**

You can use DBSLICEPARM= in numerous locations. The usual rules of option precedence apply: A table option has the highest precedence, then a LIBNAME option, and so on. SAS configuration file option has the lowest precedence because DBSLICEPARM= in any of the other locations overrides that configuration setting.

DBSLICEPARM=ALL and DBSLICEPARM=THREADED_APPS make SAS programs eligible for threaded Reads. To see whether threaded Reads are actually generated, turn on SAS tracing and run a program, as shown in this example.

```sas
options sastrace=",,t" sastraceloc=saslog nostsuffix;
proc print data=lib.dbtable(dbsliceparm=(ALL));
   where dbcol>1000;
run;
```

If you want to directly control the threading behavior, use the DBSLICE= data set option.

**Greenplum, HAWQ:** There is no default value for the maximum number of connections per table read. This value depends on the number of partitions in a table and on arguments that are used with the MOD function in a WHERE clause. For more information, see “**Autopartitioning Techniques in SAS/ACCESS**” on page 59.

**DB2 under UNIX and PC Hosts, Informix, Microsoft SQL Server, ODBC, Sybase, Sybase IQ:** The default thread number depends on whether an application passes in the number of threads (CPUCOUNT=) and whether the data type of the column that was selected for data partitioning is binary.
Examples

**Example 1: Disable Threaded Read for All SAS Users**
Here is how to use DBSLICEPARM= in a SAS configuration file entry in Windows to turn off threaded Reads for all SAS users.

-dbsliceparm NONE

**Example 2: Enable Threaded Reads for Read-Only References**
Here is how you can use DBSLICEPARM= as a z/OS invocation option to turn on threaded Reads for read-only references to DBMS tables throughout a SAS job.

sas o(dbsliceparm=ALL)

**Example 3: Increase Maximum Threads (as a SAS Global Option)**

option dbsliceparm=(threaded_apps,3);

**Example 4: Enable Threaded Reads for References Using a Particular Libref**
You can use DBSLICEPARM= as a LIBNAME option to turn on threaded Reads for read-only table references that use this particular libref, as shown in this example.

libname dblib oracle user=myusr1 password=mypwd1 dbsliceparm=ALL;

**Example 5: Enable Threaded Reads as a Table-Level Option**
Here is how to use DBSLICEPARM= as a table-level option to turn on threaded Reads for this particular table, requesting up to four connections.

proc reg SIMPLE;
   data=dblib.customers (dbsliceparm=‘all,4’);
   var age weight;
   where years_active>1;
run;

---

**DBSRVTP= SAS System Option**

Specifies whether SAS/ACCESS engines hold or block the originating client while making performance-critical calls to the database.

- **Valid in:** SAS invocation
- **Category:** Communications: Networking and Encryption
- **Default:** NONE
- **Data source:** DB2 under UNIX and PC Hosts, Informix, Netezza, ODBC, OLE DB, Oracle, Sybase, Teradata

**Syntax**

```
DBSRVTP='ALL' | 'NONE' | '(engine-name(s))'
```
Syntax Description

**ALL**
indicates that SAS does not use any blocking operations for all underlying SAS/ACCESS engines that support this option.

**NONE**
indicates that SAS uses standard blocking operations for all SAS/ACCESS engines.

**engine-name(s)**
indicates that SAS does not use any blocking operations for the specified SAS/ACCESS engines. You can specify one or more engine names. If you specify more than one engine name, separate them with blank spaces and enclose the list in parentheses.

- db2 (under UNIX and PC Hosts only)
- informix
- netezza
- odbc (indicates that SAS uses non-blocking operations for SAS/ACCESS ODBC and Microsoft SQL Server interfaces)
- oledb
- oracle
- sybase
- teradata (not supported on z/OS)

Details

This option applies only when SAS is called as a server responding to multiple clients. You can use this option to help throughput of the SAS server because it supports multiple simultaneous execution streams, if the server uses certain SAS/ACCESS interfaces. Improved throughput occurs when the underlying SAS/ACCESS engine does not hold or block the originating client. That is, any one client using a SAS/ACCESS product does not keep the SAS server from responding to other client requests. SAS/SHARE software and SAS Integration Technologies are two ways of invoking SAS as a server.

This option is a system invocation option, which means the value is set when SAS is invoked. Because the DBSRVTP= option uses multiple native threads, enabling this option uses the underlying DBMS's threading support. Some databases handle threading better than others, so you might want to invoke DBSRVTP= for some DBMSs and not others. Refer to your documentation for your DBMS for more information.

The option accepts a string where values are the engine name of a SAS/ACCESS product, ALL, or NONE. When specifying multiple values, enclose the values in quotation marks and parentheses, and separate the values with a space.

This option is applicable on all Windows platforms, AIX, Solaris, and z/OS (Oracle only). On some of these hosts, you can call SAS with the -SETJMP system option. Setting -SETJMP disables the DBSRVTP= option.

Example

Each of these examples calls SAS from the UNIX command line.

```
sas -dbsrvtp all
sas -dbsrvtp '(oracle db2)'
```
SASTRACE= SAS System Option

Generates trace information from a DBMS engine.

Valid in: configuration file, SAS invocation, OPTIONS statement
Category: Log and procedure output control: SAS log
Default: none
Data source: DB2 under UNIX and PC Hosts, DB2 under z/OS, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, Sybase, Teradata

Tip: You can also use more than one SASTRACE= option at a time (for example, SASTRACE=",,,d,d").

See: SASTRACELoc= system option, Generating Trace Information for Threaded Reads on page 56

Syntax

SASTRACE=",,,d" | ",,,d," | "d," | ",,,db" | ",,,sa" | ",,,t"

Syntax Description

',,,d'
specifies that all SQL statements that are sent to the DBMS are sent to the log. Here are the applicable statements:

- SELECT
- DELETE
- CREATE
- SYSTEM CATALOG
- DROP
- COMMIT
- INSERT
- ROLLBACK
- UPDATE

For engines that do not generate SQL statements, API calls and all parameters are sent to the log.

',,,d,' specifies that all routine calls are sent to the log. All function enters, exits, and pertinent parameters and return codes are traced when you select this option. The information varies from engine to engine, however.

This option is most useful if you have a problem and need to send a SAS log to technical support for troubleshooting.

',,,t'
specifies that all DBMS calls (such as API and client calls, connection information, column bindings, column error information, and row processing) are sent to the log. This information varies from engine to engine, however.
This option is most useful if you have a problem and need to send a SAS log to technical support for troubleshooting.

',,,db'
specifies that only a brief version of all SQL statements that the ',,,d' option normally generates are sent to the log.

',,,s'
specifies that a summary of timing information for calls made to the DBMS is sent to the log.

',,,sa'
specifies that timing information for each call that is made to the DBMS is sent to the log along with a summary.

',,t,'
specifies that all threading information is sent to the log. Here is the information that it includes:

• number of threads that are spawned
• number of observations that each thread contains
• exit code of the thread, if it fails

Details

SASTRACE= and SASTRACELOC= behavior is specific to SAS/ACCESS software. SASTRACE= is a very powerful tool to use when you want to see the commands that SAS/ACCESS sent to your DBMS. SASTRACE= output is DBMS-specific. However, most SAS/ACCESS engines show you statements like SELECT or COMMIT as the DBMS processes them for the SAS application. These details can help you manage SASTRACE= output in your DBMS.

• When using SASTRACE= on PC platforms, you must also specify SASTRACELOC=.

• Here is how to turn SAS tracing off:

  options sastrace=off;

• Log output is much easier to read if you specify NOSTSUFFIX. Because this code is entered without specifying the option, the resulting log is longer and harder to decipher.

  *Note:* NOSTSUFFIX is not supported on z/OS.

  options sastrace=',,,d' sastraceloc=saslog;
  proc print data=mydblib.snow_birthdays;
  run;

  Here is the resulting log.

  0 1349792597 sastb_next 2930 PRINT
  ORACLE_5: Prepared: 1 1349792597 sastb_next 2930 PRINT
  SELECT * FROM scott.SNOW_BIRTHDAYS 2 1349792597 sastb_next 2930 PRINT
  3 1349792597 sastb_next 2930 PRINT
  16 proce print data=mydblib.snow_birthdays; run;
  4 1349792597 sastb_next 2930 PRINT
  ORACLE_6: Executed: 5 1349792597 sastb_next 2930 PRINT
  Prepared statement ORACLE_5 6 1349792597 sastb_next 2930 PRINT
  7 1349792597 sastb_next 2930 PRINT

  Use NOSTSUFFIX to make the log file much easier to read.
Here is the resulting log.

ORACLE_1: Prepared:
SELECT * FROM scott.SNOW_BIRTHDAYS;
12 proc print data=mydblib.snow_birthdays; run;
ORACLE_2: Executed:
Prepared statement ORACLE_1

Examples

**Example 1: Use SQL Trace ',,,d'**
These examples use NOSTSUFFIX and SASTRACELOC=SASLOG.

data work.winter_birthdays;
   input empid birthdat date9. lastname $18.;
   format birthdat date9.;
datalines;
678999  28DEC1966  PAVEO           JULIANA    3451
456788  12JAN1977  SHIPTON          TIFFANY   3468
890123  20FEB1973  THORSTAD         EDVARD    3329
;
run;

Examples are based on this data set.

options sastrace=',,d' sastraceloc=saslog nostsuffix;
libname mydblib oracle user=myusr1 password=mypwd1 schema=bday_data;
data mydblib.snow_birthdays;
   set work.winter_birthdays;
run;
libname mydblib clear;

Output for this ',,,d' example is written to the SAS log, as specified in the SASTRACELOC=SASLOG option.
Example 2: Use Log Trace 'd'

options sastrace=',d' sastraceloc=saslog nostsuffix;
libname mydblib oracle user=myusr1 password=XXXXX schema=bday_data;
data mydblib.snow_birthdays;
   set work.winter_birthdays;
run;
libname mydblib clear;

NOTE: Libref MYDBLIB has been deassigned.
set work.winter_birthdays;
run;
libname mydblib clear;

Output is written to the SAS log, as specified in the SASTRACELOC=SASLOG option.

Output 13.2  SAS Log Output from the SASTRACE= ' , , d , ' System Option

NOTE: Libref MYDBLIB was successfully assigned as follows:
  Engine:        ORACLE
  Physical Name: lupin

NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.

NOTE: There were 3 observations read from the data set WORK.WINTER_BIRTHDAYS.
ORACLE: orforc()
ORACLE: orflush()
NOTE: The data set MYDBLIB.SNOW_BIRTHDAYS has 3 observations and 3 variables.
ACCESS ENGINE: Enter yoeclos
ACCESS ENGINE: Entering dbiclose
ORACLE: orclose()
ORACLE: orforc()
ORACLE: orflush()
ACCESS ENGINE: DBICLOSE open_id 0, connect_id 2
ACCESS ENGINE: Exiting dbiclose with rc=0X00000000
ACCESS ENGINE: Entering DBIDCON
ORACLE: ordcon
ACCESS ENGINE: Physical disconnect on id = 2
ACCESS ENGINE: Exiting DBIDCON with rc=0X00000000, rc2=0X00000000
ACCESS ENGINE: Exit yoeclos with rc=0X00000000
NOTE: DATA statement used (Total process time):
   real time           0.21 seconds
   cpu time            0.06 seconds
ACCESS ENGINE: Entering DBIDCON
ORACLE: ordcon
ACCESS ENGINE: Physical disconnect on id = 1
ACCESS ENGINE: Exiting DBIDCON with rc=0X00000000, rc2=0X00000000
89 libname mydblib clear;
NOTE: Libref MYDBLIB has been deassigned.

Example 3: Use DBMS Trace 'd,'
options sastrace='d,' sastraceloc=saslog nostsuffix;
libname mydblib oracle user=myusr1 password=mypwd1 schema=bday_data;
data mydblib.snow_birthdays;
   set work.winter_birthdays;
run;
libname mydblib clear;

Output is written to the SAS log, as specified in the SASTRACELOC=SASLOG option.
Output 13.3  SAS Log Output from the SASTRACE='d,' System Option

Example 4: Use Brief SQL Trace ',,,db'

options sastrace=',,db' sastraceloc=saslog nostsuffix;
libname mydblib oracle user=myusr1 password=mypwd1 path=mysrv1;
data mydblib.employee1;
   set mydblib.employee;
run;

Output is written to the SAS log, as specified in the SASTRACELOC=SASLOG option.
Output 13.4  SAS Log Output from the SASTRACE= ',,,db' System Option

| ORACLE_23: Prepared: on connection 2   |
| SELECT * FROM EMPLOYEE               |
| ORACLE_24: Prepared: on connection 3   |
| SELECT * FROM EMPLOYEE1              |
| NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables. |
| ORACLE_25: Executed: on connection 4  |
| CREATE TABLE EMPLOYEE1 (NAME VARCHAR2 (20), ID NUMBER (5), CITY VARCHAR2 (15), SALARY NUMBER, DEPT NUMBER (5)) |
| ORACLE_26: Executed: on connection 2  |
| SELECT statement ORACLE_23           |
| ORACLE_27: Prepared: on connection 4  |
| INSERT INTO EMPLOYEE1 (NAME, ID, CITY, SALARY, DEPT) VALUES (:NAME, :ID, :CITY, :SALARY, :DEPT) |
| **NOTE**: ORACLE_27 on connection 4   |
| The Execute statements associated with this Insert statement are suppressed due to SASTRACE brief setting-SASTRACE=',,,bd'. Remove the 'b' to obtain full trace. |
| NOTE: There were 17 observations read from the data set MYDBLIB.EMPLOYEE. |

Example 5: Use Time Trace ',,,s'

options sastrace=',,,s' sastraceloc=saslog nostsuffix;
libname mydblib oracle user=myusr1 password=mypwd1 schema=bday_data;
data mydblib.snow_birthdays;
   set work.winter_birthdays;
run;
libname mydblib clear;

Output is written to the SAS log, as specified in the SASTRACELOC=SASLOG option.

Output 13.5  SAS Log Output from the SASTRACE= ',,,s' System Option

| 118 options sastrace=',,,s' sastraceloc=saslog nostsuffix; |
| 119 libname mydblib oracle user=myusr1 password=mypwd1 schema=bday_data; |
| NOTE: Libref MYDBLIB was successfully assigned as follows: |
| Engine: ORACLE |
| Physical Name: lupin |
| 120 data mydblib.snow_birthdays; |
| 121 set work.winter_birthdays; |
| 122 run; |
| NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables. |
| NOTE: There were 3 observations read from the data set WORK.WINTER_BIRTHDAYS. |
| NOTE: The data set MYDBLIB.SNOW_BIRTHDAYS has 3 observations and 3 variables. |
| Summary Statistics for ORACLE are: |
| Total SQL execution seconds were: 0.127079 |
| Total SQL prepare seconds were: 0.004404 |
| Total SQL row insert seconds were: 0.004735 |
| Total seconds used by the ORACLE ACCESS engine were 0.141860 |
| NOTE: DATA statement used (Total process time): |
| real time 0.21 seconds |
| cpu time 0.04 seconds |
| 123 libname mydblib clear; |
| NOTE: Libref MYDBLIB has been deassigned. |

Example 6: Use Time All Trace ',,,sa'

options sastraces=',,,sa' sastraceloc=saslog nostsuffix;
libname mydblib oracle user=myusr1 password=mypwd1 schema=bday_data;
data mydblib.snow_birthdays;
  set work.winter_birthdays;
run;
libname mydblib clear;

Output is written to the SAS log, as specified in the SASTRACELOC=SASLOG option.

Output 13.6  SAS Log Output from the SASTRACE=','sa' System Option

Example 7: Use Threaded Trace ',,t,'
options sastrace=',,t,' sastraceloc=saslog nostsuffix;
libname mydblib oracle user=myusr1 password=mypwd1 schema=bday_data;
data mydblib.snow_birthdays(DBTYPE=(empid'number(10');
  set work.winter_birthdays;
run;
proc print data=mydblib.snow_birthdays(dbsliceparm=(all,3));
run;

Output is written to the SAS log, as specified in the SASTRACELOC=SASLOG option.
### SASLogOutput from the SASTRACE= ',,t,' System Option

```
165  options sastrace=',,t,' sastraceloc=saslog nostsuffix;
166  data mydblib.snow_birthdays(DBTYPE=(empid='number(10)'));
167     set work.winter_birthdays;
168  run;
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 3 observations read from the data set WORK.WINTER_BIRTHDAYS.
NOTE: The data set MYDBLIB.SNOW_BIRTHDAYS has 3 observations and 3 variables.
NOTE: DATA statement used (Total process time):
     real time        0.21 seconds
     cpu time         0.06 seconds
169  proc print data=mydblib.snow_birthdays(dbsliceparm=(all,3));
170  run;
ORACLE:  DBSLICEPARM option set and 3 threads were requested
ORACLE:  No application input on number of threads.
ORACLE:  Thread 1 contains 1 obs.
ORACLE:  Thread 2 contains 0 obs.
ORACLE:  Thread 3 contains 2 obs.
ORACLE:  Threaded read enabled. Number of threads created: 3
NOTE: There were 3 observations read from the data set MYDBLIB.SNOW_BaaaaaAYS.
NOTE: PROCEDURE PRINT used (Total process time):
     real time        1.12 seconds
     cpu time         0.17 seconds
```

### SASTRACELOC= SAS System Option

Specifies the location where SASTRACE= information should be printed.

- **Valid in:** configuration file, SAS invocation, OPTIONS statement
- **Category:** SAS log and procedure output: SAS log
- **Default:** stdout
- **Data source:** DB2 under PC and UNIX hosts, Informix, Microsoft SQL Server, MySQL, Netezza, ODBC, OLE DB, Oracle, Sybase, Teradata
- **See:** SASTRACE= system option

#### Syntax

```
SASTRACELOC= stdout | SASLOG | FILE 'path-and-filename'
```

#### Required Arguments

- **stdout**
  - specifies the default output location for your operating environment.

- **SASLOG**
  - specifies that trace messages should be printed to the SAS log.

- **FILE='path-and-filename'**
  - specifies that trace messages should be printed to a filename that you provide. If you do not provide a path, the file is generated in the current working directory.
Details

SASTRACELOC= lets you specify where to put the trace messages that SASTRACE= generates.

This option and its values might differ for each host.

Example: Specify a Location for the Trace Log

This example writes trace information to the TRACE.LOG file in the work directory on the C drive on a PC platform.

```plaintext
options sastrace=',,,d' sastraceloc=file 'c:\work\trace.log';
```

SQLGENERATION= System Option

Specifies whether and when SAS procedures generate SQL for in-database processing of source data.

- **Valid in:** configuration file, SAS invocation, OPTIONS statement, SAS System Options window
- **Categories:** Data Access, System Administration: Performance
- **Default:** (NONE DBMS='TERADATA DB2 ORACLE NETEZZA ASTER GREENPLM HADOOP SAPHANA IMPALA HAWQ')
- **Restrictions:** Parentheses are required when this option value contains multiple keywords. The maximum length of the option value is 4096 characters. For DBMS= and EXCLUDEDB= values, the maximum length of an engine name is eight characters. For the EXCLUDEPROC= value, the maximum length of a procedure name is 16 characters. An engine can appear only once, and a procedure can appear only once for a given engine. Not all procedures support SQL generation for in-database processing for every engine type. If you specify a setting that is not supported, an error message indicates the level of SQL generation that is not supported. The procedure can then reset to the default so that source table records can be read and processed within SAS. If this is not possible, the procedure ends and sets SYSERR= as needed. If you are using the Metadata LIBNAME Engine, the only valid SQLGENERATION= modifiers are NONE and DBMS. The engine ignores the DBMS=, EXCLUDEDB=, and EXCLUDEPROC= modifiers.

- **Requirement:** You must specify NONE or DBMS as the primary state.
- **Interactions:** Use this option with such procedures as PROC FREQ to indicate that SQL is generated for in-database processing of DBMS tables through supported SAS/ACCESS engines. You can specify different SQLGENERATION= values for the DATA= and OUT= data sets by using different LIBNAME statements for each of these data sets.
- **Data source:** Aster, DB2 under UNIX and PC Hosts, DB2 under z/OS, Greenplum, Hadoop, HAWQ, Impala, Netezza, Oracle, SAP HANA, Teradata

**Note:** Support for Impala and HAWQ was added in the third maintenance release for SAS 9.4.

**Tip:** After you set a required value (primary state), you can specify optional values (modifiers).
See:  SQLGENERATION= LIBNAME option (includes examples)
“Running In-Database Procedures” in SAS In-Database Products: User's Guide

Syntax

SQLGENERATION=<(DBMS=<DBMS='engine1 engine2...enginen'>
<EXCLUDEDB=engine | 'engine1 ...enginen'>
<EXCLUDEPROC="engine='proc1...procn' enginen='proc1...procn' ">)>
SQLGENERATION=" "

Required Arguments

NONE
prevents those SAS procedures that are enabled for in-database processing from
generating SQL for in-database processing. This is a primary state.

DBMS
allows SAS procedures that are enabled for in-database processing to generate SQL
for in-database processing of DBMS tables through supported SAS/ACCESS
engines. This is a primary state.

Note:  As a best practice, run as many calculations in-database as possible.
Processing that is run in-database generally results in better performance.

" "
resets the value to the default that was shipped.

Optional Arguments

DBMS=’engine1…engineN’
specifies one or more SAS/ACCESS engines. It modifies the primary state.

EXCLUDEDB=’engine | ’engine1…engineN’
prevents SAS procedures from generating SQL for in-database processing for one or
more specified SAS/ACCESS engines.

EXCLUDEPROC=’engine=’proc1…procN’ engineN=’proc1…procN’ ”
prevents one or more specified SAS procedures from generating SQL for in-database
processing for one or more specified SAS/ACCESS engines.

Details

Here is how SAS/ACCESS handles precedence between the LIBNAME and system
option.
Table 13.1  Precedence of Values for SQLGENERATION= LIBNAME and System Options

<table>
<thead>
<tr>
<th>LIBNAME Option</th>
<th>PROC EXCLUDE on System Option?</th>
<th>Engine Specified on System Option</th>
<th>Resulting Value</th>
<th>From (option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not set</td>
<td>yes</td>
<td>NONE</td>
<td>NONE</td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS</td>
<td>EXCLUDEPROC</td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td></td>
<td>NONE</td>
<td>NONE</td>
<td>LIBNAME</td>
</tr>
<tr>
<td>DBMS</td>
<td></td>
<td>NONE</td>
<td>EXCLUDEPROC</td>
<td>system</td>
</tr>
<tr>
<td>not set</td>
<td>no</td>
<td>NONE</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBMS</td>
<td>DBMS</td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td></td>
<td>NONE</td>
<td>NONE</td>
<td>LIBNAME</td>
</tr>
<tr>
<td>DBMS</td>
<td></td>
<td>NONE</td>
<td>DBMS</td>
<td></td>
</tr>
</tbody>
</table>

Example

Here is the default that is shipped with the product.

```sas
options sqlgeneration='';
proc options option=sqlgeneration
run;
```

SAS procedures generate SQL for in-database processing for all databases except DB2 in this example.

```sas
options sqlgeneration='';
options sqlgeneration=(DBMS EXCLUDEDB='DB2');
proc options option=sqlgeneration;
run;
```

In this example, in-database processing occurs only for Teradata. SAS procedures that are run on other databases do not generate SQL for in-database processing.

```sas
options sqlgeneration='';
options SQLGENERATION=(NONE DBMS='Teradata');
proc options option=sqlgeneration;
run;
```

For this example, SAS procedures generate SQL for Teradata and Oracle in-database processing. However, no SQL is generated for PROC1 and PROC2 in Oracle.

```sas
options sqlgeneration='';
options SQLGENERATION = (NONE DBMS='Teradata Oracle'
  EXCLUDEPROC="oracle='proc1 proc2'");
**SQLMAPPUTTO= System Option**

Specifies whether the PUT function is mapped to the SAS_PUT() function for a database, possible also where the SAS_PUT() function is mapped.

- **Valid in:** configuration file, SAS invocation, OPTIONS statement
- **Category:** Files: SAS Files
- **Default:** SAS_PUT
- **Data source:** Aster, DB2 under UNIX and PC Hosts, Greenplum, Netezza, Teradata
- **See:** SQL_FUNCTIONS= LIBNAME option, SAS In-Database Products: User's Guide

**Syntax**

`SQLMAPPUTTO=NONE | SAS_PUT | (database.SAS_PUT)`

**Syntax Description**

- **NONE**
  - specifies to PROC SQL that no PUT mapping is to occur.

- **SAS_PUT**
  - specifies that the PUT function be mapped to the SAS_PUT() function.

- **database.SAS_PUT**
  - specifies the database name.

**TIP**

- It is not necessary that the format definitions and the SAS_PUT() function reside in the same database as the one that contains the data that you want to format. You can use the `database.SAS_PUT` argument to specify the database where the format definitions and the SAS_PUT() function have been published.

- **TIP**
  - The database name can be a multilevel name and it can include blanks.

**Requirement**

- If you specify a database name, you must enclose the entire argument in parentheses.

**Details**

The format publishing macros deploy or publish, the PUT function implementation to the database as a new function named SAS_PUT(). The format publishing macros also publish both user-defined formats and formats that SAS supplies that you create using PROC FORMAT. The SAS_PUT() function supports the use of SAS formats. You can use it in SQL queries that SAS submits to the database so that the entire SQL query can be processed inside the database. You can also use it in conjunction with in-database procedures.

You can use this option with the SQLREDUCEPUT=, SQLREDUCEPUTOBS, and SQLREDUCEPUTVALUES= system options. For more information about these options, see *SAS SQL Procedure User's Guide*.  

```sql
proc options option=sqlgeneration;
run;
```
VALIDVARNANE= SAS System Option

Controls the type of SAS variable names that can be used or created during a SAS session.

**Valid in:** configuration file, SAS invocation, OPTIONS statement, SAS System Options window

**Category:** Files: SAS Files

**Default:** V7

**See:** Introduction to SAS/ACCESS Naming

### Syntax

VALIDVARNANE=V7 | UPCASE | ANY

### Required Arguments

**VALIDVARNANE=V7**

- Indicates that a DBMS column name is changed to a valid SAS name, following these rules. This is the default value for SAS 7 and later.
  - Up to 32 mixed-case alphanumeric characters are allowed.
  - Names must begin with an alphabetic character or an underscore.
  - Invalid characters are changed to underscores.
  - Any column name that is not unique when it is normalized is made unique by appending a counter (0,1,2,...) to the name.

**VALIDVARNANE=UPCASE**

- Indicates that a DBMS column name is changed to a valid SAS name as described in VALIDVARNANE=V7 except that variable names are in uppercase.

**VALIDVARNANE=ANY**

- Allows any characters in DBMS column names to appear as valid characters in SAS variable names. Symbols, such as the equal sign (=) and the asterisk (*), must be contained in a ‘variable-name’ construct. You must use ANY whenever you want to read DBMS column names that do not follow the SAS naming conventions.

### Details

The VALIDVARNANE= system option is supported for all DBMSs that support the SQL pass-through facility. You can set this option on start-up or in an OPTIONS statement, and the option value is used in the call to the SQL procedure. Alternatively, you can specify the VALIDVARNANE= option in the CONNECT statement.

### Examples

**Example 1: Rename Columns during View Creation**

This example shows how the SQL pass-through facility works with VALIDVARNANE=V7.

```plaintext
options validvarname=v7;
proc sql;
```
connect to oracle (user=myusr1 pass=mypwd1);
create view myview as
  select amount_b, amount_s
  from connection to oracle
  (select "Amount Budgeted$", "Amount Spent$"
      from mytable);
quit;
proc contents data=myview;
runt;

Output from this example would show that "Amount Budgeted$" becomes AMOUNT_B and "Amount Spent$" becomes AMOUNT_S.

**Example 2: Pass VALIDVARNAME= as a Connection Option**

This example shows how you can pass VALIDVARNAME= as a connection option in the CONNECT statement in the SQL procedure.

```sql
proc sql;
  drop view work.TLV1;
  connect to vertica (validvarname=v7 server=myserver port=5433
user=myuserid password=mypwd database=test);
  exec("drop table vartb") by vertica;
  exec("create table vartb ( c1234567890 int, c123456789012345678 float ) ") by vertica;
  exec("insert into vartb values (123, 3.14159)") by vertica;
  create view tlv1 as
    select * from connection to vertica (select * from vartb);
quit;
```
Chapter 14
SQL Pass-Through Facility for Relational Databases

About SQL Procedure Interactions

Overview of SQL Procedure Interactions with SAS/ACCESS ........................................ 471
SQL Pass-Through Facility ......................................................... 471

Syntax: SQL Pass-Through Facility for Relational Databases ........................................ 472
Overview ................................................................. 472
Return Codes ............................................................. 472

Dictionary ........................................................... 473
CONNECT Statement ................................................. 473
CONNECTION TO Component ........................................ 477
DISCONNECT Statement ............................................... 480
EXECUTE Statement ..................................................... 480

About SQL Procedure Interactions

Overview of SQL Procedure Interactions with SAS/ACCESS

The SQL procedure implements Structured Query Language (SQL) for SAS software. For details about PROC SQL, see the SAS SQL Procedure User's Guide. Here is how you can use SAS/ACCESS software for relational databases for PROC SQL interactions.

- You can assign a libref to a DBMS using the SAS/ACCESS LIBNAME statement and reference the new libref in a PROC SQL statement to query, update, or delete DBMS data. (See LIBNAME Statement for Relational Databases on page 96.)
- You can embed LIBNAME information in a PROC SQL view and then automatically connect to the DBMS every time the PROC SQL view is processed. (See SQL Views with Embedded LIBNAME Statements on page 99.)
- You can send DBMS-specific SQL statements directly to a DBMS using an extension to PROC SQL called the SQL pass-through facility. (See Syntax for the SQL Pass-Through Facility for Relational Databases on page 472.)

SQL Pass-Through Facility

The SQL pass-through facility uses SAS/ACCESS to connect to a DBMS and to send statements directly to the DBMS for execution. As an alternative to the SAS/ACCESS LIBNAME statement, this facility lets you use the SQL syntax of your DBMS. It supports any SQL that is not ANSI-standard that your DBMS supports.
Not all SAS/ACCESS interfaces support this feature, however. To determine whether it is available in your environment, see SAS/ACCESS features by host on page 77.

Here are the tasks that you can complete by using the SQL pass-through facility.

• Establish and terminate connections with a DBMS using its CONNECT and DISCONNECT statements.
• Send dynamic, non-query, DBMS-specific SQL statements to a DBMS using its EXECUTE statement.
• Retrieve data directly from a DBMS using its CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement.

You can use SQL pass-through facility statements in a PROC SQL query, or you can store them in an SQL view. When you create an SQL view, any arguments that you specify in the CONNECT statement are stored with the view. Therefore, when you use the view in a SAS program, SAS can establish the appropriate connection to the DBMS.

For DBMS-specific details about the SQL pass-through facility, see the reference section for your SAS/ACCESS interface.

Syntax: SQL Pass-Through Facility for Relational Databases

Overview

The syntax section presents the syntax for the SQL pass-through facility statements and the CONNECTION TO component. For DBMS-specific details, see the DBMS-specific reference section for your SAS/ACCESS interface.

PROC SQL <option(s)>;
CONNECT TO dbms-name <AS alias>
<(<database-connection-arguments> <connect-statement-arguments> )>;
DISCONNECT FROM dbms-name | alias;
EXECUTE (dbms-specific-SQL-statement) BY dbms-name | alias;
SELECT column-list FROM CONNECTION TO dbms-name | alias (dbms-query)

Return Codes

As you use the PROC SQL statements that are available in the SQL pass-through facility, any error return codes and error messages are written to the SAS log. These codes and messages are available to you through these SAS macro variables:

SQLXRC
contains the DBMS return code that identifies the DBMS error.

SQLXMSG
contains descriptive information about the DBMS error that the DBMS generates.

The contents of the SQLXRC and SQLXMSG macro variables are printed in the SAS log using the %PUT macro. They are reset after each SQL pass-through facility statement has been executed. For details about these return codes, see “Macro Variables for Relational Databases” on page 445.
Dictionary

CONNECT Statement

Establishes a connection with the DBMS

Valid in: PROC SQL steps (when accessing DBMS data using SAS/ACCESS software)

Syntax

CONNECT TO dbms-name <AS alias> <(connect-statement-arguments)> <(database-connection-arguments)>;
CONNECT USING libref <AS alias>;

Required Arguments
dbms-name
specifies the DBMS to which you want to connect. You must specify the DBMS name for your SAS/ACCESS interface. See the SQL pass-through section in the DBMS-specific reference section for your SAS/ACCESS interface.

libref
specifies the libref for which a DBMS connection has already been established through the LIBNAME statement.

Optional Arguments
AS alias
specifies an alias for the connection that has 1 to 32 characters. The AS keyword must precede alias. If you do not specify an alias, the DBMS name is used as the name of the SQL pass-through connection. Some DBMSs allow more than one connection. You can use the AS clause to name connections so that you can refer to them later.

connect-statement-arguments
specifies values for arguments that indicate whether you can make multiple connections, shared or unique connections, and so on, to the database. With these arguments, the SQL pass-through facility can use some of the connection management features of the LIBNAME statement or of SAS system options. Although these arguments are optional, you must enclose them in parentheses if you include any:

• CONNECTION=
• CONNECTION_GROUP=
• DBCONINIT=
• DBCONTERM=
• DBGEN_NAME=
• DBMAX_TEXT=
• DBPROMPT=
• DEFER=
• VALIDVARNAME=

Note: In addition to the arguments that are listed here, several other LIBNAME or system options are available for use with the CONNECT statement. For options that are available for your SAS/ACCESS interface, see the SQL pass-through facility section for your interface in the DBMS-specific reference section. When used with the SQL pass-through facility CONNECT statement, these options have the same effect as they do in a LIBNAME statement.

CONNECTION= SHARED | GLOBAL
indicates whether multiple CONNECT statements for a DBMS can use the same connection.

The CONNECTION= option enables you to control the number of connections, and therefore transactions, that your SAS/ACCESS engine executes and supports for each SQL pass-through CONNECT statement.

When CONNECTION=GLOBAL, multiple CONNECT statements that use identical values for CONNECTION=, CONNECTION_GROUP=, DBCONINIT=, DBCONTERM=, and any database connection arguments can share the same connection to the DBMS.

When CONNECTION=SHARED, the CONNECT statement makes one connection to the DBMS. Only SQL pass-through statements that use this alias share the connection. SHARED is the default value for CONNECTION=.

In this example, the two CONNECT statements share the same connection to the DBMS because CONNECTION=GLOBAL. Only the first CONNECT statement actually makes the connection to the DBMS. The last DISCONNECT statement is the only statement that disconnects from the DBMS.

proc sql;
/*...SQL Pass-Through statements referring to mydbone...*/
connect to oracle as mydbone
  (user=myusr1 pw=mypwd1 path='mysrv1' connection=global);
/*...SQL Pass-Through statements referring to mydbtwo...*/
connect to oracle as mydbtwo
  (user=myusr1 pw=mypwd1 path='mysrv1' connection=global);
disconnect from mydbone;
disconnect from mydbtwo;
quit;

CONNECTION_GROUP=connection-group-name
specifies a connection that can be shared among several CONNECT statements in the SQL pass-through facility.

Default: none

By specifying the name of a connection group, you can share one DBMS connection among several CONNECT statements. The connection to the DBMS can be shared only if each CONNECT statement specifies the same CONNECTION_GROUP= value and specifies identical DBMS connection arguments.

When CONNECTION_GROUP= is specified, it implies that the value of the CONNECTION= option is GLOBAL.
DBCONINIT='DBMS-user-command'
specifies a user-defined initialization command to be executed immediately after the connection to the DBMS.

You can specify any DBMS command that can be passed by the SAS/ACCESS engine to the DBMS and that does not return a result set or output parameters. The command executes immediately after the DBMS connection is established successfully. If the command fails, a disconnect occurs, and the CONNECT statement fails. You must specify the command as a single, quoted string, unless it is an environment variable.

DBCONTERM='DBMS-user-command'
specifies a user-defined termination command to be executed before the disconnect from the DBMS that occurs with the DISCONNECT statement.

**Default:** none

The termination command that you select can be a script, stored procedure, or any DBMS SQL language statement that might provide additional control over the interaction between the SAS/ACCESS engine and the DBMS. You can specify any valid DBMS command that can be passed by the SAS/ACCESS engine to the DBMS and that does not return a result set or output parameters. The command executes immediately before SAS terminates each connection to the DBMS. If the command fails, SAS provides a warning message but the disconnect still occurs. You must specify the command as a quoted string.

DBGEN_NAME= DBMS | SAS
specifies whether to automatically rename DBMS columns containing characters that SAS does not allow, such as $, to valid SAS variable names.

DBMAX_TEXT=integer
determines the length of any very long DBMS character data type that is read into SAS or written from SAS when using a SAS/ACCESS engine. This option applies to reading, appending, and updating rows in an existing table. It does not apply when you are creating a table.

Examples of a long DBMS data type are the Sybase TEXT data type and the Oracle LONG RAW data type.

DBPROMPT=YES | NO
specifies whether SAS displays a window that prompts the user to enter DBMS connection information before connecting to the DBMS.

**Default:** NO

**Interaction:** DEFER= LIBNAME option

If you specify DBPROMPT=YES, SAS displays a window that interactively prompts you for the DBMS connection arguments when the CONNECT statement is executed. Therefore, it is not necessary to provide connection arguments with the CONNECT statement. If you do specify connection arguments with the CONNECT statement and you specify DBPROMPT=YES, the connection argument values are displayed in the window. These values can be overridden interactively.

If you specify DBPROMPT=NO, SAS does not display the prompting window.

The DBPROMPT= option interacts with the DEFER= LIBNAME option to determine when the prompt window appears. If DEFER=NO, the DBPROMPT window appears when the CONNECT statement is executed. If DEFER=YES, the DBPROMPT window appears the first time a pass-through statement is executed. The DEFER= option normally defaults to NO. The option defaults to
YES if DBPROMPT=YES. You can override this default by explicitly setting DEFER=NO.

**DEFER=NO | YES**
determines when the connection to the DBMS occurs.

*Default:* NO

If DEFER=YES, the connection to the DBMS occurs when the first SQL pass-through statement is executed. If DEFER=NO, the connection to the DBMS occurs when the CONNECT statement occurs.

**VALIDVARMNAME=V7**
indicates that only SAS 7 variable names are considered valid. Specify this connection argument if you want the SQL pass-through facility to operate in SAS 7 compatibility mode.

By default, DBMS column names are changed to valid SAS names, following these rules:

- Up to 32 mixed-case alphanumeric characters are allowed.
- Names must begin with an alphabetic character or an underscore.
- Characters that are not permitted are changed to underscores.
- Any column name that is not unique when it is normalized is made unique by appending a counter (0,1,2,...) to the name.

When VALIDVARMNAME=V7 is specified, the SAS/ACCESS engine for the DBMS truncates column names to eight characters, as it does in SAS 7. If required, numbers are appended to the ends of the truncated names to make them unique. Setting this option overrides the value of the SAS system option VALIDVARMNAME= during (and only during) the SQL pass-through connection.

This example shows how the SQL pass-through facility uses VALIDVARMNAME=V7 as a connection argument. Using this option causes the output to show the DBMS column "Amount Budgeted$" as AMOUNT_B and "Amount Spent$" as AMOUNT_S.

```sql
proc sql;
connect to oracle (user=myusr1 password=mypwd1 validvarmname=v7)
create view budget2000 as
  select amount_b, amount_s
  from connection to oracle
    (select "Amount Budgeted$", "Amount Spent$"
     from annual_budget);
quit;
```

```sql
proc contents data=budget2000;run;
```

For this example, if you omit VALIDVARMNAME=V7 as a connection argument, you must add it in an OPTIONS= statement in order for PROC CONTENTS to work.

```sql
options validvarmname=v7;
proc contents data=budget2000;run;
```

So using it as a connection argument saves you coding later.

**database-connection-arguments**
specifies values for DBMS-specific arguments that PROC SQL needs to connect to the DBMS. Though they are optional for most databases, you must enclose them in
parentheses if you include any. For information about these arguments, see the
DBMS-specific reference section for your SAS/ACCESS interface.

Details

The CONNECT statement establishes a connection with the DBMS. You establish a
connection to send DBMS-specific SQL statements to the DBMS or to retrieve DBMS
data. The connection remains in effect until you issue a DISCONNECT statement or
terminate the SQL procedure.

Follow these steps to connect to a DBMS using the SQL pass-through facility.

1. Initiate a PROC SQL step.

2. Use the SQL pass-through facility CONNECT statement, identify the DBMS (such
   as Oracle or DB2), and assign an (optional) alias.

3. Specify any attributes for the connection such as SHARED or UNIQUE.

4. Specify any arguments that are needed to connect to the database.

The CONNECT statement is optional for some DBMSs. However, if you do not specify
it, the default values for all database connection arguments are used.

A CONNECT TO statement results in a new database connection each time you use it. A
CONNECT USING statement uses the existing connection that is established in the
LIBNAME statement for the specified libref. If you are accessing tables in the same
library multiple times, CONNECT USING statements might improve performance.

When you specify a CONNECT USING statement, any connection options that are
included in the LIBNAME statement are not used. Instead, the default values for all
database connection arguments for your DBMS are used.

Any return code or message that the DBMS generates is available in the SQLXRC and
SQLXMSG macro variables after the statement executes. For information about these
macro variables, see “Macro Variables for Relational Databases” on page 445.

Example: CONNECT Statement

This example connects to a Sybase server and assigns the alias SYBCON1 to it. Sybase
is a case-sensitive database, so database objects are in uppercase, as they were created.

```
proc sql;
connect to sybase as sybcon1
  (server=MYSRV1 database=PERSONNEL user=MYUSR1
   password=MYPWD1 connection=global);
%put &sqlxmsg &sqlxrc;
```

You might be able to omit the CONNECT statement and implicitly connect to a database
by using default settings. For details, see the DBMS-specific reference section for your
SAS/ACCESS interface.

---

**CONNECTION TO Component**

Retrieves and uses DBMS data in a PROC SQL query or view

**Valid in:** PROC SQL step SELECT statements (when accessing DBMS data using
SAS/ACCESS software)
Syntax

CONNECTION TO dbms-name | alias | (dbms-query)

Required Arguments

dbms-name

identifies the DBMS to which you direct the DBMS-specific SQL statement. See the SQL pass-through section in the DBMS-specific reference section for your SAS/ACCESS interface.

alias

specifies an alias, if one was defined in the CONNECT statement.

(dbms-query)

specifies the query that you are sending to the DBMS. The query can use any DBMS-specific SQL statement or syntax that is valid for the DBMS.

You must specify a query argument in the CONNECTION TO component, and the query must be enclosed in parentheses. The query is passed to the DBMS exactly as you enter it. Therefore, if your DBMS is case sensitive, you must use the correct case for DBMS object names.

On some DBMSs, the dbms-query argument can be a DBMS stored procedure. However, stored procedures with output parameters are not supported in the SQL pass-through facility. Furthermore, if the stored procedure contains more than one query, only the first query is processed.

Details

The CONNECTION TO component specifies the DBMS connection that you want to use or that you want to create (if you have omitted the CONNECT statement). CONNECTION TO then enables you to retrieve DBMS data directly through a PROC SQL query.

You use the CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement:

PROC SQL;
    SELECT column-list
    FROM CONNECTION TO dbms-name (dbms-query) other optional PROC SQL clauses
QUIT;

You can use CONNECTION TO in any FROM clause, including those in nested queries—that is, subqueries.

You can store an SQL pass-through facility query in an SQL view and then use that view in SAS programs. When you create an SQL view, any options that you specify in the corresponding CONNECT statement are stored too. So when the SQL view is used in a SAS program, SAS can establish the appropriate connection to the DBMS.

On many relational databases, you can issue a CONNECTION TO component in a PROC SQL SELECT statement directly without first connecting to a DBMS. (See CONNECTION statement on page 473.) If you omit the CONNECT statement, an implicit connection is performed by using default values for all database connection arguments. This automatic connection occurs at the first PROC SQL SELECT statement that contains a CONNECTION TO component. For details, see the SQL pass-through section in the DBMS-specific reference section for your SAS/ACCESS interface.

Because relational databases and SAS have different naming conventions, some DBMS column names might be changed when you retrieve DBMS data through the

Examples

Example 1: Send an Oracle SQL Query to the Oracle Database
After you connect to a DBMS by using either the CONNECT statement or implicit default settings, you can send a DBMS-specific SQL query to the DBMS using the CONNECTION TO component. You specify the columns that you want to retrieve, identify your DBMS type, and issue your query by using the SQL syntax of your DBMS.

This example sends an Oracle SQL query (highlighted below) to the Oracle database for processing. The results from the Oracle SQL query serve as a virtual table for the PROC SQL FROM clause. In this example, MYCON is a connection alias.

```
proc sql;
connect to oracle as mycon (user=myusr1
   password=mypwd1 path='mysrv1');
%put &sqlxmsg;
select *
from connection to mycon
   (select empid, lastname, firstname,
    hiredate, salary
    from employees where
    hiredate>='31-DEC-88');
%put &sqlxmsg;   
disconnect from mycon;
quit;
```

The SAS %PUT macro displays the &SQLXMSG macro variable for error codes and information from the DBMS. For details, see “Macro Variables for Relational Databases” on page 445.

Example 2: Name and Store the Query as an SQL View
This example gives the query a name and stores it as the SQL view samples.

```
libname samples 'SAS-library';
proc sql;
connect to oracle as mycon (user=myusr1
   password=mypwd1 path='mysrv1');
%put &sqlxmsg;
create view samples.hires88 as
   select *
from connection to mycon
   (select empid, lastname, firstname,
    hiredate, salary
    from employees where
    hiredate>='31-DEC-88');
%put &sqlxmsg;   
disconnect from mycon;
quit;
```
DISCONNECT Statement
Terminates the connection to the DBMS

Valid in: PROC SQL steps (when accessing DBMS data using SAS/ACCESS software)

Syntax
DISCONNECT FROM dbms-name | alias

Required Arguments

*dbms-name*
specifies the DBMS from which you want to disconnect. You must either specify the DBMS name for your SAS/ACCESS interface or use an alias in this statement. See the LIBNAME section in the DBMS-specific reference section for your SAS/ACCESS interface. If you used the CONNECT statement to connect to the DBMS, the DBMS name or alias in the DISCONNECT statement must match what you specified in the CONNECT statement.

*alias*
specifies an alias that was defined in the CONNECT statement.

Details
The DISCONNECT statement ends the connection with the DBMS. If you do not include the DISCONNECT statement, SAS performs an implicit DISCONNECT when PROC SQL terminates. The SQL procedure continues to execute until you submit a QUIT statement, another SAS procedure, or a DATA step.

Any return code or message that is generated by the DBMS is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See “Macro Variables for Relational Databases” on page 445 for more information about these macro variables.

Example
To exit the SQL pass-through facility, use the facilities DISCONNECT statement and then QUIT the PROC SQL statement. This example disconnects the user from a DB2 database with the alias DBCON1 and terminates the SQL procedure.

```sas
proc sql;
  connect to db2 as dbcon1 (ssid=db2a);
  ...more SAS statements...
  disconnect from dbcon1;
  quit;
```

EXECUTE Statement
Sends DBMS-specific, non-query SQL statements to the DBMS

Valid in: PROC SQL steps (when accessing DBMS data using SAS/ACCESS software)
Syntax
EXECUTE (dbms-specific-sql-statement) BY dbms-name | alias;

Required Arguments
(dbms-specific-sql-statement)
a dynamic non-query, DBMS-specific SQL statement. This argument is required and must be enclosed in parentheses. The SQL statement might be case sensitive, depending on your DBMS, and it is passed to the DBMS exactly as you enter it.

On some DBMSs, this argument can be a DBMS stored procedure. However, stored procedures with output parameters are not supported in the SQL pass-through facility. Furthermore, if the stored procedure contains more than one query, only the first query is processed.

Any return code or message that is generated by the DBMS is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See Macro Variables for Relational Databases on page 445 for more information about these macro variables.

dbms-name
specifies the DBMS to which you direct the DBMS-specific SQL statement. The keyword BY must appear before the dbms-name argument. You must either specify the DBMS name for your SAS/ACCESS interface or use an alias. See the SQL pass-through section in the DBMS-specific reference section for your SAS/ACCESS interface.

alias
specifies an alias that was defined in the CONNECT statement. (You cannot use an alias if the CONNECT statement was omitted.)

Details
Overview
The EXECUTE statement sends dynamic non-query, DBMS-specific SQL statements to the DBMS and processes those statements.

In some SAS/ACCESS interfaces, you can issue an EXECUTE statement directly without first explicitly connecting to a DBMS. (See CONNECT statement on page 473.) If you omit the CONNECT statement, an implicit connection is performed by using default values for all database connection arguments when the first EXECUTE statement is passed to the DBMS. For details, see the SQL pass-through section in the DBMS-specific reference section for your SAS/ACCESS interface.

The EXECUTE statement cannot be stored as part of an SQL pass-through facility query in a PROC SQL view.

Useful Statements to Include in EXECUTE Statements
You can pass these statements to the DBMS by using the SQL pass-through facility EXECUTE statement.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>creates a DBMS table, view, index, or other DBMS object, depending on how the statement is specified.</td>
</tr>
</tbody>
</table>
### Statement Description

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>deletes rows from a DBMS table.</td>
</tr>
<tr>
<td>DROP</td>
<td>deletes a DBMS table, view, or other DBMS object, depending on how the statement is specified.</td>
</tr>
<tr>
<td>GRANT</td>
<td>gives users the authority to access or modify objects such as tables or views.</td>
</tr>
<tr>
<td>INSERT</td>
<td>adds rows to a DBMS table.</td>
</tr>
<tr>
<td>REVOKE</td>
<td>revokes the access or modification privileges that were given to users by the GRANT statement.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>modifies the data in one column of a row in a DBMS table.</td>
</tr>
</tbody>
</table>

For more information and restrictions on these and other SQL statements, see the SQL documentation for your DBMS.
Part 3

DBMS-Specific Reference

Chapter 15
SAS/ACCESS Interface to Aster ........................................... 485

Chapter 16
SAS/ACCESS Interface to DB2 for UNIX and PC Hosts ................. 503

Chapter 17
SAS/ACCESS Interface to DB2 for z/OS ................................. 531

Chapter 18
SAS/ACCESS Interface to Greenplum .................................. 583

Chapter 19
SAS/ACCESS Interface to Hadoop ......................................... 603

Chapter 20
SAS/ACCESS Interface to HAWQ .......................................... 629

Chapter 21
SAS/ACCESS Interface to Impala ......................................... 649

Chapter 22
SAS/ACCESS Interface to Informix ...................................... 663

Chapter 23
SAS/ACCESS Interface to Microsoft SQL Server ...................... 681

Chapter 24
SAS/ACCESS Interface to MySQL ....................................... 697

Chapter 25
SAS/ACCESS Interface to Netezza ....................................... 711

Chapter 26
SAS/ACCESS Interface to ODBC ......................................... 727

Chapter 27
SAS/ACCESS Interface to OLE DB ....................................... 755
Chapter 28
SAS/ACCESS Interface to Oracle ............................... 779

Chapter 29
SAS/ACCESS Interface to PostgreSQL ....................... 811

Chapter 30
SAS/ACCESS Interface to SAP HANA .......................... 827

Chapter 31
SAS/ACCESS Interface to Sybase .............................. 849

Chapter 32
SAS/ACCESS Interface to Sybase IQ .......................... 873

Chapter 33
SAS/ACCESS Interface to Teradata ........................... 891

Chapter 34
SAS/ACCESS Interface to Vertica .............................. 939
Chapter 15
SAS/ACCESS Interface to Aster

Introduction to SAS/ACCESS Interface to Aster ................................................. 486
LIBNAME Statement Specifics for Aster ............................................................ 486
  Overview ........................................................................................................ 486
  Arguments .................................................................................................... 486
  Aster LIBNAME Statement Examples ............................................................ 489
Data Set Options for Aster ............................................................................... 490
SQL Pass-Through Facility Specifics for Aster .................................................... 492
  Key Information ............................................................................................ 492
  CONNECT Statement Example ...................................................................... 492
  Special Catalog Queries .............................................................................. 493
Autopartitioning Scheme for Aster ................................................................... 494
  Overview ........................................................................................................ 494
  Autopartitioning Restrictions ...................................................................... 494
  Nullable Columns .......................................................................................... 494
  Using WHERE Clauses ................................................................................ 494
  Using DBSLICE= .......................................................................................... 495
  Using DBSLICE=} ....................................................................................... 495
Passing SAS Functions to Aster ....................................................................... 495
Passing Joins to Aster ...................................................................................... 496
Bulk Loading and Unloading for Aster .............................................................. 497
  Loading ......................................................................................................... 497
  Data Set Options with Bulk Loading .......................................................... 497
  Examples ....................................................................................................... 497
  Unloading ...................................................................................................... 498
  Example ....................................................................................................... 498
Naming Conventions for Aster ......................................................................... 499
Data Types for Aster ......................................................................................... 499
  Overview ....................................................................................................... 499
  Supported Aster Data Types ...................................................................... 499
  LIBNAME Statement Data Conversions ....................................................... 500
Introduction to SAS/ACCESS Interface to Aster

For available SAS/ACCESS features, see Aster supported features on page 78. For more information about Aster, see your Aster documentation.

LIBNAME Statement Specifics for Aster

Overview

This section describes the LIBNAME statement options that SAS/ACCESS Interface to Aster supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Aster.

\[ \text{LIBNAME } \text{libref} \text{ aster } <\text{connection-options}> <\text{LIBNAME-options}>; \]

Arguments

\textit{libref}

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

\textit{aster}

specifies the SAS/ACCESS engine name for the Aster interface.

\textit{connection-options}

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to the Aster database in several ways. Specify only one of these methods for each connection because they are mutually exclusive.

- \texttt{SERVER=, DATABASE=, PORT=, USER=, PASSWORD=}
- \texttt{DSN=, USER=, PASSWORD=}
- \texttt{NOPROMPT=}
- \texttt{PROMPT=}
- \texttt{REQUIRED=}

Here is how these options are defined.

\texttt{SERVER=<server-name>}

specifies the host name or IP address where the Aster database is running. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

\texttt{DATABASE=<database-name>}

specifies the Aster database that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.
PORT=port
 specifies the port number that is used to connect to the specified Aster database. If you do not specify a port, the default port 2406 is used.

USER='Aster-user-name'
 specifies the Aster user name (also called the user ID) that you use to connect to your database. If the user name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

PASSWORD='Aster-password'
 Alias: PASS=, PW=, PWD=
 specifies the password that is associated with your Aster user ID. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

DSN='Aster-data-source'
 specifies the configured Aster ODBC data source to which you want to connect. Use this option if you have existing Aster ODBC data sources that are configured on your client. This method requires additional setup, either through the ODBC Administrator control panel on Windows platforms or through the odbc.ini file on UNIX platforms. So it is recommended that you use this connection method only if you have existing, functioning data sources that have been defined.

Requirement: If you use this method, you just specify the user ID and password in the LIBNAME statement, even if these are already specified in the ODBC data source.

NOPROMPT='Aster-ODBC-connection-options'
 specifies connection options for your data source or database. Separate multiple options with a semicolon. If you do not specify enough correct connection options, an error is returned. No dialog box is displayed to help you with the connection string.

PROMPT='Aster-ODBC-connection-options'
 specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMSG macro variable. PROMPT= does not immediately try to connect to the DBMS. It instead displays a dialog box that contains the values that you entered in the PROMPT= connection string. You can edit values or enter additional values in any field before you connect to the data source. This option is not supported on UNIX platforms.

REQUIRED='Aster-ODBC-connection-options'
 specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMSG macro variable. If you do not specify enough correct connection options, a dialog box prompts you for the connection options. REQUIRED= lets you modify only required fields in the dialog box. This option is not supported on UNIX platforms.

LIBNAME -options
define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Aster with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.
Table 15.1 SAS/ACCESS LIBNAME Options for Aster

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>operation-specific</td>
</tr>
<tr>
<td>BULKUNLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>UNIQUE</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIMENSION=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>LOGIN_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>PARTITION_KEY=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for Aster”</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>see “Naming Conventions for Aster”</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SQLGENERATION=</td>
<td>DBMS</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>USE_ODBC_CL=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Aster LIBNAME Statement Examples**

In this first example, SERVER=, DATABASE=, USER=, and PASSWORD= are the connection options.
LIBNAME mydblib ASTER SERVER=mysrv1 DATABASE=test
    USER=myusr1 PASSWORD=mypwd1;

PROC Print DATA=mydblib.customers;
    WHERE state='CA';
run;

In this next example, the DSN= option, the USER= option, and the PASSWORD=
option are connection options. The Aster data source is configured in the ODBC
Administrator Control Panel on Windows platforms. It is also configured in the odbc.ini
file or a similarly named configuration file on UNIX platforms.

LIBNAME mydblib aster dsn=aster user=myusr1 password=mypwd1;

PROC Print DATA=mydblib.customers;
    WHERE state='CA';
run;

Here is how you can use the NOPROMPT= option.

libname x aster NOPROMPT="dsn=aster;";
libname x aster NOPROMPT="DRIVER=aster; server=192.168.28.100;
    uid=username; pwd=password; database=asterdb";

This example uses the PROMPT= option. Blanks are also passed down as part of the
connection options. Therefore, the specified value must immediately follow the
semicolon.

libname x aster PROMPT="DRIVER=aster;";

The REQUIRED= option is used in this example. If you enter all needed connection
options, REQUIRED= does not prompt you for any input.

libname x aster REQUIRED="DRIVER=aster; server=192.168.28.100;
    uid=username; pwd=password; database=asterdb ;";

As shown above, when asterdb (contains a trailing blank) is specified as the database
instead of asterdb (no trailing blank), this error results:

ERROR: CLI error trying to establish connection:
    ERROR: Database asterdb does not exist.

---

Data Set Options for Aster

All SAS/ACCESS data set options in this table are supported for Aster. Default values
are provided where applicable. For details, see Data Set Options for Relational
Databases on page 241.

Table 15.2  SAS/ACCESS Data Set Options for Aster

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DATAFILE_PATH=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DBNAME=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_DELIMITER=</td>
<td>\t (the tab symbol)</td>
</tr>
<tr>
<td>BL_ESCAPE= on page 280</td>
<td>\ \</td>
</tr>
<tr>
<td>BL_HOST=</td>
<td>none</td>
</tr>
<tr>
<td>BL_OPTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>BL_PATH=</td>
<td>none</td>
</tr>
<tr>
<td>BL_QUOTE= on page 306</td>
<td>&quot;</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>BULKUNLOAD=</td>
<td>LIBNAME option value</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for Aster”</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for Aster”</td>
</tr>
<tr>
<td>DIMENSION=</td>
<td>NO</td>
</tr>
</tbody>
</table>
Option | Default Value
--- | ---
DISTRIBUTE_ON= | none
ERRLIMIT= | 1
IGNORE_READ_ONLY_COLUMNS= | NO
INSERTBUFF= | LIBNAME option setting
NULLCHAR= | SAS
NULLCHARVAL= | A blank character
PARTITION_KEY= | none
PRESENCE_COL_NAMES= | LIBNAME option setting
QUERY_TIMEOUT= | LIBNAME option setting
READBUFF= | LIBNAME option setting
SASDATEFMT= | none
SCHEMA= | LIBNAME option setting

### SQL Pass-Through Facility Specifics for Aster

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Aster interface.

- The `dbms-name` is **ASTER**.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to Aster. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default `ASTER` alias is used.
- The CONNECT statement `database-connection-arguments` are identical to its `LIBNAME` connection options.

**CONNECT Statement Example**

This example uses the DBCON alias to connect to **myserver1** the Aster database and execute a query. The connection alias is optional.
proc sql;
  connect to aster as dbcon
    (server=mysrv1 database=test user=myusr1 password=mypwd1);
select * from connection to dbcon
  (select * from customers WHERE customer like '1%');
quit;

Special Catalog Queries

SAS/ACCESS Interface to Aster supports the following special queries. You can use the queries to call the ODBC-style catalog function application programming interfaces (APIs). Here is the general format of the special queries:

Aster::SQLAPI'parameter-1', 'parameter-n'

Aster:: is required to distinguish special queries from regular queries. Aster:: is not case sensitive.

SQLAPI
is the specific API that is being called. SQLAPI is not case sensitive.

'parameter n'
is a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters, and the underscore represents any single character. To use either character as a literal value, you can use the backslash character (\) to escape the match characters. For example, this call to SQL Tables usually matches table names such as myatest and my_test:

    select * from connection to aster (ASTER::SQLTables "test","","my_test");

Use the escape character to search only for the my_test table.

    select * from connection to aster (ASTER::SQLTables "test","","my\_test");

SAS/ACCESS Interface to Aster supports these special queries.

ASTER::SQLTables <'Catalog', 'Schema', 'Table-name', 'Type'>
returns a list of all tables that match the specified arguments. If you do not specify any arguments, all accessible table names and information are returned.

ASTER::SQLColumns <'Catalog', 'Schema', 'Table-name', 'Column-name'>
returns a list of all columns that match the specified arguments. If you do not specify any argument, all accessible column names and information are returned.

ASTER::SQLPrimaryKeys <'Catalog', 'Schema', 'Table-name', 'Type'>
returns a list of all columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If you do not specify any table name, this special query fails.

ASTER::SQLStatistics <'Catalog', 'Schema', 'Table-name'>
returns a list of the statistics for the specified table name. You set options SQL_INDEX_ALL and SQL_ENSURE in the SQLStatistics API call. If you do not specify a table name argument, this special query fails.
ASTER::SQLGetTypeInfo returns information about the data types that the Aster database supports.

ASTER::SQLTablePrivileges<"Catalog", "Schema", "Table-name"> returns a list of all tables and associated privileges that match the specified arguments. If no arguments are specified, all accessible table names and associated privileges are returned.

---

### Autopartitioning Scheme for Aster

#### Overview

Autopartitioning for SAS/ACCESS Interface to Aster is a modulo (MOD) function method. For general information about this feature, see "Autopartitioning Techniques in SAS/ACCESS" on page 59.

#### Autopartitioning Restrictions

SAS/ACCESS Interface to Aster places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here is how columns are partitioned.

- **SQL_INTEGER, SQL_BIT, SQL_SMALLINT, and SQL_TINYINT** columns are given preference.
- You can use **SQL_DECIMAL, SQL_DOUBLE, SQL_FLOAT, SQL_NUMERIC, and SQL_REAL** columns for partitioning under these conditions:
  - Aster supports converting these types to **SQL_INTEGER** by using the INTEGER cast function.
  - The precision minus the scale of the column is greater than 0 but less than 10; namely, \(0<(\text{precision}-\text{scale})<10\).

#### Nullable Columns

If you select a nullable column for autopartitioning, the `OR<column-name>IS NULL` SQL statement is appended at the end of the SQL code that is generated for the threaded Read. This ensures that any possible NULL values are returned in the result set. Also, if the column to be used for the partitioning is **SQL_BIT**, the number of threads are automatically changed to two, regardless of `DBSLICEPARM=` option setting.

#### Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in a WHERE clause. For example, this DATA step could not use a threaded Read to retrieve the data. All numeric columns in the table are in the WHERE clause:

```sas
DATA work.locemp;
   SET trlib.MYEMPS;
   WHERE EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```
Using DBSLICEParm=

SAS/ACCESS Interface to Aster defaults to three threads when you use autopartitioning but do not specify a maximum number of threads in to use for the threaded Read. See DBSLICEParm= LIBNAME option.

Using DBSLICE=

You might achieve the best possible performance when using threaded Reads by specifying the DBSLICE= data set option for Aster in your SAS operation. Using DBSLICE= allows connections to individual partitions so that you can configure an Aster data source for each partition. Use this option to specify both the data source and the WHERE clause for each partition.

```
proc print data=trilb.MYEMPS(DBSLICE=(DSN1='EMPNUM BETWEEN 1 AND 33'
DSN2='EMPNUM BETWEEN 34 AND 66'
DSN3='EMPNUM BETWEEN 67 AND 100'));
run;
```

Using the DATASOURCE= option is not required to use DBSLICE= option with threaded Reads.

Using DBSLICE= works well when the table that you want to read is not stored in multiple partitions. It gives you flexibility in column selection. For example, if you know that the STATE column in your employee table contains only a few distinct values, you can customize your DBSLICE= option accordingly.

```
data work.locemp;
  set trilb2.MYEMP(DBSLICE=("STATE='FL'","STATE='GA'",
"STATE='SC'","STATE='VA'","STATE='NC'"));
  where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

Passing SAS Functions to Aster

SAS/ACCESS Interface to Aster passes the following SAS functions to Aster for processing. Where the Aster function name differs from the SAS function name, the Aster name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>Aster Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>MIN</td>
</tr>
<tr>
<td>ARCOS (ACOS)</td>
<td>MINUTE (date_part)</td>
</tr>
<tr>
<td>ARSIN (ASIN)</td>
<td>MLOWCASE (lower)</td>
</tr>
<tr>
<td>ATAN</td>
<td>MOD (see note)</td>
</tr>
<tr>
<td>ATAN2</td>
<td>MONTH (date_part)</td>
</tr>
<tr>
<td>AVG</td>
<td>QTR (date_part)</td>
</tr>
<tr>
<td>BYTE (chr)</td>
<td>REPEAT</td>
</tr>
<tr>
<td>CEIL (ceiling)</td>
<td>SIGN</td>
</tr>
<tr>
<td>COALESCE</td>
<td>SIN</td>
</tr>
<tr>
<td>COS</td>
<td>SQRT</td>
</tr>
<tr>
<td>COUNT</td>
<td>STRIP (btrim)</td>
</tr>
</tbody>
</table>
DAY (date_part)    SUBSTR (substring)
EXP                SUM
FLOOR              TAN
HOUR (date_part)   TRANWRD (replace)
INDEX (strpos)     TRIMN (rtrim)
LOG (ln)           UPCODE (upper)
LOG10 (log)        YEAR (date_part)
MAX

Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Aster. Due to incompatibility in date and time functions between Aster and SAS, Aster might not process them correctly. Check your results to determine whether these functions are working as expected. For more information, see “SQL_FUNCTIONS= LIBNAME Option” on page 213.

COMPRESS (replace) ROUND
DATE (now::date)   TIME (now::time)
DATEPART (cast)    TIMEPART (cast)
DATETIME (now)     TODAY (now::date)
LENGTH             TRANSLATE

Passing Joins to Aster

For a multiple libref join to pass to Aster, all of these components of the LIBNAME statements must match exactly.

• user ID (USER=)
• password (PASSWORD=)
• server (SERVER=)
• database (DATABASE=)
• port (PORT=)
• data source (DSN=, if specified)
• SQL functions (SQL_FUNCTIONS=)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.
Bulk Loading and Unloading for Aster

Loading

Bulk loading is the fastest way to insert large numbers of rows into an Aster table. You must specify `BULKLOAD=YES` to use the bulk-load facility. The bulk-load facility uses the Aster loader client application to move data from the client to the Aster database.

Data Set Options with Bulk Loading

Here are the Aster bulk-load data set options. For detailed information about these options, see “About the Data Set Options for Relational Databases” on page 241.

- `BL_DATAFILE`
- `BL_DATAFILE_PATH`
- `BL_DBNAME`
- `BL_DELETE_DATAFILE`
- `BL_DELIMITER`
- `BL_HOST`
- `BL_OPTIONS`
- `BL_PATH`
- `BULKLOAD`
- `BULKUNLOAD`

Examples

This example shows how you can use a SAS data set, SASFLT.FLT98, to create and load a large Aster table, FLIGHTS98.

```sas
LIBNAME sasflt 'SAS-library';
LIBNAME net_air ASTER user=myusr1 pwd=mypwd1
     server=air2 database=flights dimension=yes;
PROC sql;
create table net_air.flights98
         (bulkload=YES bl_host='queen' bl_path='/home/aster_loader/'
           bl_dbname='beehive')
     as select * from sasflt.flt98;
quit;
```

You can use `BL_OPTIONS=` to pass specific Aster options to the bulk-loading process.

You can create the same table using a DATA step.

```sas
data net_air.flights98(bulkload=YES bl_host='queen'
               bl_path='/home/aster_loader/'
               bl_dbname='beehive');
set sasflt.flt98;
```
run;

You can then append the SAS data set, SASFLT.FLT98, to the existing Aster table, ALLFLIGHTS. SAS/ACCESS Interface to Aster to write data to a flat file, as specified in the BL_DATAFILE= option. Rather than deleting the data file, BL_DELETE_DATAFILE=NO causes the engine to retain it after the load completes.

PROC append base=net_air.allflights
   (BULKLOAD=YES
    BL_DATAFILE='/tmp/fltdata.dat'
    BL_HOST='queen'
    BL_PATH='/home/aster_loader/'
    BL_DBNAME='beehive'
    BL_DELETE_DATAFILE=NO )
data=sasflt.flt98;
run;

Unloading

Bulk unloading is the fastest way to insert large numbers of rows from an Aster table. To use the bulk-unloading facility, specify BULKUNLOAD=YES. (See BULKUNLOAD= on page 318.) The bulk-unloading facility uses the Aster Remote External Table interface to move data from the client to the Aster Performance Server into SAS.

Here are the Aster bulk-unloading data set options:

- BL_DATAFILE=
- BL_DELETE_DATAFILE=
- BL_DELIMITER=
- BL_USE_PIPE=
- BULKLOAD=
- BULKUNLOAD=

Example

This example shows you how you can read a large Aster table to create and populate a SAS data set:

libname db aster server ='redqueen.unx.sas.com'
   db='accesstesting' user='dbittest' password='dbigrp1' dimension=yes;
proc sql;
select * from db.employees(bulkunload=YES);
create table work.employees as select * from db.employees(bulkunload=YES
   bl_options=' ' BL_DATAFILE='c:\bl_data.dat' BL_USE_PIPE=NO
   BL_DELETE_DATAFILE=NO);
quit;

proc append base=work.employees data=db.empployees(bulkunload=YES
   bl_options=' ' BL_DATAFILE='c:\bl_data.dat' BL_USE_PIPE=NO
   BL_DELETE_DATAFILE=NO);
run;
Naming Conventions for Aster

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The Aster interface supports table names and column names that contain up to 32 characters. If column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name would result in identical column names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a name longer than 32 characters. If you have a table name that is greater than 32 characters, it is recommended that you create a table view.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Aster is not case sensitive, so all names default to lowercase.

Aster objects include tables, views, and columns. They follow these conventions.

• A name must be from 1 to 64 characters long.
• A name must begin with a letter (A through Z), diacritic marks, non-Latin characters (200–377 octal) or an underscore (_).
• To enable case sensitivity, enclose names in quotation marks. All references to quoted names must always be enclosed in quotation marks, and preserve case sensitivity.
• A name cannot begin with a _bee prefix. Leading _bee prefixes are reserved for system objects.
• A name cannot be a reserved word in Aster such as WHERE or VIEW.
• A name cannot be the same as another Aster object that has the same type.

Data Types for Aster

Overview

Every column in a table has a name and a data type. The data type tells Aster how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Aster data types and data conversions.

SAS/ACCESS Interface to Aster does not directly support TIMETZ or INTERVAL types. Any columns using these types are read into SAS as character strings.

For information about Aster data types and to determine which data types are available for your version of Aster, see your Aster documentation.

Supported Aster Data Types

Here are the data types that the Aster engine supports.
• Character data:
  CHAR(n)   VARCHAR(n)

• Numeric data:
  BIGINT     DOUBLE | DOUBLE PRECISION
  SMALLINT   REAL
  INTEGER    DECIMAL | DEC | NUMERIC | NUM

  Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

• Date, time, and timestamp data:
  DATE       TIMESTAMP
  TIME

  Note: Be aware that columns of these data types can contain data values that are out of range for SAS.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to Aster assigns to SAS variables when using the LIBNAME statement to read from an Aster table. These default formats are based on Aster column attributes.

<table>
<thead>
<tr>
<th>Aster Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>VARCHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>numeric</td>
<td>11.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.</td>
</tr>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>Aster Data Type</td>
<td>SAS Data Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME8.</td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME25.6</td>
</tr>
</tbody>
</table>

* n in Aster data types is equivalent to w in SAS formats.

This table shows the default Aster data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

**Table 15.4 LIBNAME Statement: Default Aster Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Aster Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.d</td>
<td>DECIMAL(p,s)</td>
</tr>
<tr>
<td>other numerics</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>$w$.</td>
<td>VARCHAR($n$)*</td>
</tr>
<tr>
<td>datetime formats</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>TIME</td>
</tr>
</tbody>
</table>

* n in Aster data types is equivalent to w in SAS formats.
Chapter 16
SAS/ACCESS Interface to DB2 for UNIX and PC Hosts

Introduction to SAS/ACCESS Interface to DB2 under UNIX and PC Hosts . . . . 504
LIBNAME Statement Specifics for DB2 under UNIX and PC Hosts . . . . . 504
  Overview ................................................. 504
  Arguments .................................................. 504
  DB2 under UNIX and PC Hosts LIBNAME Statement Example ............ 508
Data Set Options for DB2 under UNIX and PC Hosts .......................... 508
SQL Pass-Through Facility Specifics for DB2 under UNIX and PC Hosts . . . 511
  Key Information ........................................... 511
  Examples ................................................... 511
  Special Catalog Queries ................................... 512
Autopartitioning Scheme for DB2 under UNIX and PC Hosts ................... 513
  Overview .................................................. 513
  Autopartitioning Restrictions ................................ 513
  Nullable Columns ......................................... 513
  Using WHERE Clauses ..................................... 513
  Using DBSLICEPARM= .................................... 514
  Using DBSLICE= .......................................... 514
  Configuring DB2 EEE Nodes on Physically Partitioned Databases ....... 515
Temporary Table Support for DB2 under UNIX and PC Hosts ................... 516
Calling Stored Procedures in DB2 under UNIX and PC Hosts ................... 516
  Overview .................................................. 516
  Examples ................................................... 516
DBLOAD Procedure Specifics for DB2 under UNIX and PC Hosts ................ 518
  Key Information ........................................... 518
  Examples ................................................... 519
Passing SAS Functions to DB2 under UNIX and PC Hosts ....................... 520
Passing Joins to DB2 under UNIX and PC Hosts ................................ 521
Bulk Loading for DB2 under UNIX and PC Hosts ................................ 521
  Overview .................................................. 521
  Using the LOAD Method ................................... 522
  Data Set Options with Bulk Loading ................................ 522
  Using the IMPORT Method .................................. 522
  Using the CLI LOAD Method ................................ 523
  Capturing Bulk-Load Statistics into Macro Variables .................... 523
  Maximizing Load Performance for DB2 under UNIX and PC Hosts ........ 523
Introduction to SAS/ACCESS Interface to DB2 under UNIX and PC Hosts

For available SAS/ACCESS features, see DB2 under UNIX and PC Hosts supported features on page 78. For more information about DB2 under UNIX and PC Hosts, see your DB2 under UNIX and PC Hosts documentation.

LIBNAME Statement Specifics for DB2 under UNIX and PC Hosts

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to DB2 under UNIX and PC Hosts supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing DB2 under UNIX and PC Hosts.

\texttt{LIBNAME \textit{libref db2} \textit{<connection-options> <LIBNAME-options>};}

Arguments

\textit{libref}

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

\textit{db2}

specifies the SAS/ACCESS engine name for the DB2 under UNIX and PC Hosts interface.

\textit{connection-options}

provides connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to DB2 several ways. Specify only one of these methods for each connection because they are mutually exclusive.

- \texttt{USER=, PASSWORD=, DATASRC=}
- \texttt{COMPLETE=}
- \texttt{NOPROMPT=}
Here is how these options are defined.

**USER=**
lets you connect to a DB2 database with a user ID that is different from the default ID. USER= is optional. If you specify USER=, you must also specify PASSWORD=. If USER= is omitted, your default user ID for your operating environment is used.

**PASSWORD=**
specifies the DB2 password that is associated with your DB2 user ID. PASSWORD= is optional. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you specify USER=, you must specify PASSWORD=.

**DATASRC=**
specifies the DB2 data source or database to which you want to connect. DATASRC= is optional. If you omit it, you connect by using a default environment variable.

Alias: DATABASE=, DB=, DSN=

**COMPLETE=**
specifies connection information for your data source or database for PCs only. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable. If you do not specify enough correct connection options, you are prompted with a dialog box that displays the values from the COMPLETE= connection string. You can edit any field before you connect to the data source. This option is not available on UNIX platforms. See your DB2 documentation for more details.

**NOPROMPT=**
specifies connection information for your data source or database. Separate multiple options with a semicolon. If you do not specify enough correct connection options, an error is returned (no dialog box is displayed).

**PROMPT=**
specifies connection information for your data source or database for PCs only. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable. PROMPT= does not immediately attempt to connect to the DBMS. Instead, it displays a dialog box that contains the values that you entered in the PROMPT= connection string. You can edit values or enter additional values in any field before you connect to the data source. This option is not available on UNIX platforms.

**REQUIRED=**
specifies connection information for your data source or database for PCs only. Separate the multiple options with semicolons. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable. If you do not specify enough correct connection options, a dialog box prompts you for the connection options. REQUIRED= lets you modify only required fields in the dialog box. This option is not available on UNIX platforms.
LIBNAME-options
defines how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to DB2 under UNIX and PC Hosts, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

Table 16.1  SAS/ACCESS LIBNAME Options for DB2 under UNIX and PC Hosts

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>varies with transaction type</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>operation-specific</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows), 10,000 (when bulk loading rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>FETCH-IDENTITY=</td>
<td>NO</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>IN=</td>
<td>none</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>NONE</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>NO (see “Naming Conventions for DB2 under UNIX and PC Hosts”)</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>NO (see “Naming Conventions for DB2 under UNIX and PC Hosts”)</td>
</tr>
<tr>
<td>PRESERVE_USER=</td>
<td>NO</td>
</tr>
<tr>
<td>PROGRAM_NAME=</td>
<td>none</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>set by the user in the DB2Cli.ini file (see “Locking in the DB2 under UNIX and PC Hosts Interface”)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>your user ID</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SQLGENERATION=</td>
<td>DBMS</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
</tbody>
</table>
### DB2 under UNIX and PC Hosts LIBNAME Statement Example

In this example, the libref MyDBLib uses the DB2 engine and the NOPROMPT= option to connect to a DB2 database. PROC PRINT is used to display the contents of the DB2 table Customers.

```
libname mydblib db2
  noprompt="dsn=usrsdsn;uid=myusr1;pwd=mypwd1;";
proc print data=mydblib.customers;
  where state='CA';
run;
```

### Data Set Options for DB2 under UNIX and PC Hosts

All SAS/ACCESS data set options in this table are supported for DB2 under UNIX and PC Hosts. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>CS (see “Locking in the DB2 under UNIX and PC Hosts Interface”)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>YES</td>
</tr>
<tr>
<td>WARN_BIGINT=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>BL_ALLOW_READ_ACCESS=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_ALLOW_WRITE_ACCESS=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_CODEPAGE=</td>
<td>the code page ID for the window</td>
</tr>
<tr>
<td>BL_COPY_LOCATION=</td>
<td>none</td>
</tr>
<tr>
<td>BL_CPU_PARALLELISM=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DATA_BUFFER_SIZE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DATAFILE=</td>
<td>the current directory</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE</td>
<td>YES</td>
</tr>
<tr>
<td>BL_DISK_PARALLELISM</td>
<td>none</td>
</tr>
<tr>
<td>BL_EXCEPTION</td>
<td>none</td>
</tr>
<tr>
<td>BL_INDEXING_MODE</td>
<td>AUTOSELECT</td>
</tr>
<tr>
<td>BL_LOAD_REPLACE</td>
<td>NO</td>
</tr>
<tr>
<td>BL_LOG</td>
<td>the current directory</td>
</tr>
<tr>
<td>BL_METHOD</td>
<td>none</td>
</tr>
<tr>
<td>BL_OPTIONS</td>
<td>none</td>
</tr>
<tr>
<td>BL_PORT_MAX</td>
<td>none</td>
</tr>
<tr>
<td>BL_PORT_MIN</td>
<td>none</td>
</tr>
<tr>
<td>BL_RECOVERABLE</td>
<td>NO</td>
</tr>
<tr>
<td>BL_REMOTE_FILE</td>
<td>none</td>
</tr>
<tr>
<td>BL_SERVER_DATAFILE</td>
<td>creates a data file in the current directory or with the default file specifications (same as for BL_DATAFILE=)</td>
</tr>
<tr>
<td>BL_WARNING_COUNT</td>
<td>2147483646</td>
</tr>
<tr>
<td>BULKLOAD</td>
<td>NO</td>
</tr>
<tr>
<td>CHAR_AS_BINARY</td>
<td>NO</td>
</tr>
<tr>
<td>CURSORTYPE</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCOMMIT</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td><em>ALL</em>=YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for DB2 under UNIX and PC Hosts”</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,3</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for DB2 under UNIX and PC Hosts”</td>
</tr>
<tr>
<td>ERLLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>FETCH_IDENTITY=</td>
<td>NO</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>IN=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>
Option | Default Value  
--- | ---  
READ_ISOLATION_LEVEL= | LIBNAME option setting  
READ_LOCK_TYPE= | LIBNAME option setting  
READBUFF= | LIBNAME option setting  
SASDATEFMT= | none  
SCHEMA= | LIBNAME option setting  
UPDATE_ISOLATION_LEVEL= | LIBNAME option setting  
UPDATE_LOCK_TYPE= | LIBNAME option setting

SQL Pass-Through Facility Specifics for DB2 under UNIX and PC Hosts

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the DB2 under UNIX and PC Hosts interface.

- The *dbms-name* is **DB2**.
- The CONNECT statement is required.
- You can connect to only one DB2 database at a time. However, you can use multiple CONNECT statements to connect to multiple DB2 data sources by using the *alias* argument to distinguish your connections.
- The CONNECT statement *database-connection-arguments* are identical to its LIBNAME connection options.
- These LIBNAME options are available with the CONNECT statement.
  - **AUTOCOMMIT**=
  - **CURSOR_TYPE**=
  - **QUERY_TIMEOUT**=
  - **READBUFF**=
  - **READ_ISOLATION_LEVEL**=

**Examples**

This example connects to the SAMPLE database and sends it two EXECUTE statements to process.
proc sql;
connect to db2 (database=sample);
execute (create view
sasdemo.whotookorders as
select ordernum, takenby,
    firstname, lastname, phone
from sasdemo.orders,
sasdemo.employees
where sasdemo.orders.takenby=
sasdemo.employees.empid)
by db2;
execute (grant select on
sasdemo.whotookorders to myusr1)
by db2;
disconnect from db2;
quit;

This example connects to the SAMPLE database by using an alias (DB1) and performs a query, shown in italic type, on the SASDEMO.CUSTOMERS table.

proc sql;
connect to db2 as db1 (database=sample);
select *
from connection to db1
(select * from sasdemo.customers
    where customer like '1%');
disconnect from db1;
quit;

Special Catalog Queries

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts supports the following special queries. You can use the queries to call the ODBC-style catalog function application programming interfaces (APIs). Here is the general format of these queries:

DB2::SQLAPI "parameter 1","parameter n"

DB2::
is required to distinguish special queries from regular queries.

SQLAPI
is the specific API that is being called. Neither DB2:: nor SQLAPI are case sensitive.

"parameter n"
is a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters, and the underscore represents any single character. To use either character as a literal value, you can use the backslash character (\) to escape the match characters. For example, this call to SQLTables usually matches table names such as mytest and my_test:

select * from connection to db2 (DB2::SQLTables "test","my_test");

Use the escape character to search only for the my_test table:

select * from connection to db2 (DB2::SQLTables "test","my\_test");
SAS/ACCESS Interface to DB2 under UNIX and PC Hosts supports these special queries:

DB2::SQLDataSources
   returns a list of database aliases that have been cataloged on the DB2 client.

DB2::SQLDBMSInfo
   returns information about the DBMS server and version. It returns one row with two
columns that describe the DBMS name (such as DB2/NT) and version (such as 8.2).

---

**Autopartitioning Scheme for DB2 under UNIX and PC Hosts**

**Overview**

Autopartitioning for SAS/ACCESS Interface to DB2 for UNIX and PC Hosts is a
modulo (MOD) function method. For general information about this feature, see
“Autopartitioning Techniques in SAS/ACCESS” on page 59.

**Autopartitioning Restrictions**

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts places additional restrictions
on the columns that you can use for the partitioning column during the autopartitioning
phase. Here is how columns are partitioned.

- INTEGER and SMALLINT columns are given preference.
- You can use other DB2 numeric columns for partitioning as long as the precision
  minus the scale of the column is between 0 and 10—namely, 0<\(\text{precision-scale}\)<10.

**Nullable Columns**

If you select a nullable column for autopartitioning, the `OR<column-name> IS NULL`
SQL statement is appended at the end of the SQL code that is generated for the threaded
Reads. This ensures that any possible NULL values are returned in the result set.

**Using WHERE Clauses**

Autopartitioning does not select a column to be the partitioning column if it appears in a
SAS WHERE clause. For example, this DATA step cannot use a threaded Read to
retrieve data because all numeric columns in the table (see the table definition in “Using
DBSLICE=” are in the WHERE clause:

```
data work.locemp;
set trlib.MYEMPS;
where EMPNUM<=30 and ISTENURE=0 and
   SALARY<=35000 and NUMCLASS>2;
run;
```
Using **DBSLICEPARM=**

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts defaults to three threads when you use autopartitioning. However, do not specify a maximum number of threads for the threaded Read in the **DBSLICEPARM= LIBNAME** option.

Using **DBSLICE=**

You might achieve the best possible performance when using threaded Reads by specifying the **DBSLICE= data set option** for DB2 in your SAS operation. This is especially true if your DB2 data is evenly distributed across multiple partitions in a DB2 Enterprise Extended Edition (EEE) database system. When you create a DB2 table under the DB2 EEE model, you can specify the partitioning key that you want to use by appending the clause **PARTITIONING KEY(column-name)** to your CREATE TABLE statement. Here is how you can accomplish this by using the LIBNAME option, **DBCREATE_TABLE_OPTS=**, within the SAS environment.

```
/*points to a triple node server*/
libname trlib2 db2 user=db2user pw="db2pwd" datasrc=sample3c
DBCREATE_TABLE_OPTS='PARTITIONING KEY(EMPNUM);

proc datasets library=trlib;
 delete MYEMPS1;run;

data trlib.myemps(drop=morf whatstate
 DBTYPE=(HIREDATE="date" SALARY="numeric(8,2)"
   NUMCLASS="smallint" GENDER="char(1)" ISTENURE="numeric(1)" STATE="char(2)"
   EMPNUM="int NOT NULL Primary Key");
format HIREDATE mmddyy10.;
do EMPNUM=1 to 100;
   morf=mod(EMPNUM,2)+1;
   if(morf eq 1) then
      GENDER='F';
   else
      GENDER='M';
   salary=(ranuni(0)*5000);
   HIREDATE=int(ranuni(13131)*3650);
   whatstate=int(EMPNUM/5);
   if(whatstate eq 1) then
      STATE='FL';
   if(whatstate eq 2) then
      STATE='GA';
   if(whatstate eq 3) then
      STATE='SC';
   if(whatstate eq 4) then
      STATE='VA';
   else
      state='NC';
   ISTENURE=mod(EMPNUM,2);
   NUMCLASS=int(EMPNUM/5)+2;
   output;
end;
run;
```
After the table MYEMPS is created on this three-node database, a third of the rows reside on each of the three nodes.

Optimization of the threaded Read against this partitioned table depends on the location of the DB2 partitions. If the DB2 partitions are on the same machine, you can use DBSLICE= with the DB2 NODENUMBER function in the WHERE clause:

```plaintext
proc print data=trlib2.MYEMPS(DBSLICE="NODENUMBER(EMPNO)=0"
   "NODENUMBER(EMPNO)=1" "NODENUMBER(EMPNO)=2");
run;
```

If the DB2 partitions reside on different physical machines, you can usually obtain the best results by using the DBSLICE= option with the SERVER= syntax in addition to the DB2 NODENUMBER function in the WHERE clause.

In the next example, DBSLICE= contains specific partitioning information for DB2. Also, Sample3a, Sample3b, and Sample3c are DB2 database aliases that point to individual DB2 EEE database nodes that exist on separate physical machines. For more information about the configuration of these nodes, see “Configuring DB2 EEE Nodes on Physically Partitioned Databases” on page 515.

```plaintext
proc print data=trlib2.MYEMPS(DBSLICE=(sample3a="NODENUMBER(EMPNO)=0"
   sample3b="NODENUMBER(EMPNO)=1" sample3c="NODENUMBER(EMPNO)=2");
run;
```

NODENUMBER is not required to use threaded Reads for SAS/ACCESS Interface to DB2 under UNIX and PC Hosts. The methods and examples described in DBSLICE= on page 347 work well in cases where the table that you want to read is not stored in multiple partitions to DB2. These methods also give you full control over which column is used to execute the threaded Read. For example, if the STATE column in your employee table contains only a few distinct values, you can modify your DBSLICE= clause accordingly:

```plaintext
data work.locemp;
set trlib2.MYEMPS (DBSLICE="STATE='GA'"
   "STATE='SC'" "STATE='VA'" "STATE='NC'");
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

**Configuring DB2 EEE Nodes on Physically Partitioned Databases**

Assuming that the database SAMPLE is partitioned across three different machines, you can create a database alias for it at each node from the DB2 Command Line Processor by issuing these commands:

```plaintext
catalog tcpip node node1 remote <hostname> server 50000
catalog tcpip node node2 remote <hostname> server 50000
catalog tcpip node node3 remote <hostname> server 50000
catalog database sample as samplea at node node1
catalog database sample as sampleb at node node2
catalog database sample as samplec at node node3
```

This enables SAS/ACCESS Interface to DB2 to access the data for the SAMPLE table directly from each node. For more information about configuring DB2 EEE to use multiple physical partitions, see the *DB2 Administrator's Guide*.  

[Autopartitioning Scheme for DB2 under UNIX and PC Hosts](#)
Temporary Table Support for DB2 under UNIX and PC Hosts

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

Calling Stored Procedures in DB2 under UNIX and PC Hosts

Overview

A stored procedure is one or more SQL statements or supported third-generation languages (3GLs, such as C) statements that are compiled into a single procedure that exists in DB2. Stored procedures might contain static (hardcoded) SQL statements. Static SQL is optimized better for some DBMS operations. In a carefully managed DBMS environment, programmers and database administrators can know the exact SQL to execute.

SAS usually generates SQL dynamically. However, database administrators can encode static SQL in a stored procedure and therefore restrict SAS users to a tightly controlled interface. When you use a stored procedure call, you must specify a schema.

SAS/ACCESS support for stored procedure includes passing input parameters, retrieving output parameters into SAS macro variables, and retrieving the result set into a SAS table. Although DB2 stored procedures can return multiple result sets, SAS/ACCESS Interface to DB2 under UNIX and PC Hosts can retrieve only a single result set.

You can call stored procedures only from PROC SQL.

Examples

Example 1: Specify a Basic Call

Use CALL statement syntax to call a stored procedure.

```
call "schema".stored_proc
```

The simplest way to call a stored procedure is to use the EXECUTE statement in PROC SQL. In this example, you execute STORED_PROC by using a CALL statement. SAS does not capture the result set.

```
proc sql;
connect to db2 (db=sample uid= pwd=);
execute {call "schema".stored_proc};
quit;
```

Example 2: Specify One Input Parameter That Returns a Result Set

You can also return a result set. In this example, MYPROC3 is executed using a CALL statement and returns a result set.

```
proc sql;
```
Example 3: Specify Three Output Parameters
The CALL statement syntax supports passing of parameters. You can specify such input parameters as numeric constants, character constants, or a null value. You can also pass input parameters by using SAS macro variable references. To capture the value of an output parameter, a SAS macro variable reference is required. This example uses a constant (1), an input/output parameter (:INOUT), and an output parameters (:OUT). Not only is the result set returned to the SAS results table, the SAS macro variables INOUT and OUT capture the parameter outputs.

```
proc sql;
connect to db2 (db=sample uid= pwd=);
%let INOUT=2;
create table sasresults as select * from connection to db2
   (call "schema".stored_proc (1,:INOUT,:OUT));
quit;
```

Example 4: Specify Three Different Parameters
The CALL statement syntax supports passing of parameters. To capture the value of an output parameter, a SAS macro variable reference is required. This example uses three output parameters (:p1, :p2, :p3:) and displays the value of each.

```
proc sql;
connect to db2 (db=sample uid= pwd=);
execute (call MYSCHEMA.MYPROC(:p1,:p2,:p3)) by db2;
%put &p1 &p2 &p3 /* display values of output parameters */
```

Example 5: Pass a NULL Parameter
In these calls, NULL is passed as the parameter to the DB2 stored procedure.

- null string literals in the call
  
  - call proc(''
  
  - call proc(""

- literal period or literal NULL in the call
  
  - call proc(.)
  - call proc(NULL)

- SAS macro variable set to NULL string
  
  - %let charparm=;
  - call proc(:charparm)

- SAS macro variable set to period (SAS numeric value is missing)
  
  - %let numparm=.;
  - call proc(:numparm)

Only the literal period and the literal NULL work generically for both DB2 character parameters and DB2 numeric parameters. For example, a DB2 numeric parameter would reject "" and %let numparm=.; would not pass a DB2 NULL for a DB2 character parameter. As a literal, a period passes NULL for both numeric and character parameters. However, it constitutes a NULL only for a DB2 numeric parameter when it is in a SAS macro variable.
You cannot pass NULL parameters by omitting the argument. For example, you cannot use this call to pass three NULL parameters.

call proc(,,)

You could use this call instead.

call proc(NULL,NULL,NULL)

**Example 6: Specify a Schema**

Use standard CALL statement syntax to execute a stored procedure that exists in another schema, as shown in this example.

```sql
proc sql;
    connect to db2 (db=sample uid= pwd=);
    execute (call OTHERSCHEMA.MYPROC1(:p1));
quit;
```

If the schema is in mixed case or lowercase, enclose the schema name in double quotation marks.

```sql
proc sql;
    connect to db2 (db=sample uid= pwd=);
    execute (call "OTHERSCHEMA".MYPROC1(:p1));
quit;
```

**Example 7: Execute Remote Stored Procedures**

If the stored procedure exists on a different DB2 instance, specify it with a valid three-part name.

```sql
select * from connection
proc sql;
    connect to db2 (db=sample uid= pwd=);
    to db2 {call MYSCHEMA.MYPROC1.prod5(:p1)};
quit;
```

---

**DBLOAD Procedure Specifics for DB2 under UNIX and PC Hosts**

**Key Information**

For general information about this feature, see Appendix 3, “DBLOAD Procedure,” on page 985.

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts supports all DBLOAD procedure statements in batch mode. Here are the DBLOAD procedure specifics for the DB2 under UNIX and PC Hosts interface.

- DBMS= value is DB2.
- Here are the database description statements that PROC DBLOAD uses:

  ```
  IN= '<database-name>';
  ```

  specifies the name of the database in which you want to store the new DB2 table. The IN= statement is required and must immediately follow the PROC
DBLOAD statement. The database-name is limited to eight characters. DATABASE= is an alias for the IN= statement.

The database that you specify must already exist. If the database name contains the _, $, @, or # special character, you must enclose it in quotation marks. DB2 recommends against using special characters in database names, however.

USER= <user name>; lets you connect to a DB2 database with a user ID that is different from the default login ID.

USER= is optional in SAS/ACCESS Interface to DB2 under UNIX and PC Hosts. If you specify USER=, you must also specify PASSWORD= . If USER= is omitted, your default user ID is used.

PASSWORD= <password>; specifies the password that is associated with your user ID.

PASSWORD= is optional in SAS/ACCESS Interface to DB2 under UNIX and PC Hosts because users have default user IDs. If you specify USER=, however, you must specify PASSWORD=.

If you do not wish to enter your DB2 password in uncoded text on this statement, see PROC PWENCODE in Base SAS Procedures Guide for a method to encode it.

• Here is the TABLE= statement:

TABLE= <schema-name.table-name>; identifies the DB2 table or DB2 view that you want to use to create an access descriptor. The table-name is limited to 18 characters. If you use quotation marks, the name is case sensitive. The TABLE= statement is required.

The schema-name is a person's name or group ID that is associated with the DB2 table. The schema name is limited to eight characters.

• Here is the NULLS statement.

NULLS variable-identifier-1 =Y|N|D < . . . variable-identifier-n =Y|N|D >; lets you specify whether the DB2 columns that are associated with the listed SAS variables allow NULL values. By default, all columns accept NULL values.

The NULLS statement accepts any one of these values.

Y specifies that the column accepts NULL values. This is the default.

N specifies that the column does not accept NULL values.

D specifies that the column is defined as NOT NULL WITH DEFAULT.

Examples

This example creates a new DB2 table, SASDEMO.EXCHANGE, from the MYDDBLIB.RATEOFEX data file. You must be granted the appropriate privileges in order to create new DB2 tables or views.

```
proc dbload dbms=db2 data=mydblib.rateofex;
in='sample';
user='myusr1';
password=mypwd1;
```
```sas
table=sasdemo.exchange;
   rename fgnindol=fgnindollars 4=dollarsinfgn;
nulls updated=n fgnindollars=n
dollarsinfgn=n country=n;
load;
run;

This example sends only a DB2 SQL GRANT statement to the SAMPLE database and
does not create a new table. Therefore, the TABLE= and LOAD statements are omitted.

proc dbload dbms=db2;
   in='sample';
   sql grant select on sasdemo.exchange
to myusr1;
run;
```

---

## Passing SAS Functions to DB2 under UNIX and PC Hosts

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts passes the following SAS functions to DB2 for processing if the DBMS driver or client that you are using supports this function. Where the DB2 function name differs from the SAS function name, the DB2 name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>DB2 Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>MIN</td>
</tr>
<tr>
<td>ARCOS (ACOS)</td>
<td>MINUTE</td>
</tr>
<tr>
<td>ARSIN (ASIN)</td>
<td>MOD (see note)</td>
</tr>
<tr>
<td>ATAN</td>
<td>MONTH</td>
</tr>
<tr>
<td>AVG</td>
<td>QTR (QUARTER)</td>
</tr>
<tr>
<td>BYTE (CHAR)</td>
<td>REPEAT</td>
</tr>
<tr>
<td>CEIL (CEILING)</td>
<td>SECOND</td>
</tr>
<tr>
<td>COS</td>
<td>SIGN</td>
</tr>
<tr>
<td>COSH</td>
<td>SIN</td>
</tr>
<tr>
<td>COT</td>
<td>SINH</td>
</tr>
<tr>
<td>COUNT (COUNT_BIG)</td>
<td>SQRT</td>
</tr>
<tr>
<td>DAY (DAYOFMONTH)</td>
<td>STRIP (RTRIM, LTRIM)</td>
</tr>
<tr>
<td>EXP</td>
<td>SUBSTR (SUBSTRING)</td>
</tr>
<tr>
<td>FLOOR</td>
<td>SUM</td>
</tr>
<tr>
<td>HOUR</td>
<td>TAN</td>
</tr>
<tr>
<td>INDEX (LOCATE)</td>
<td>TANH</td>
</tr>
<tr>
<td>LENGTH</td>
<td>TRANWRD (REPLACE)</td>
</tr>
<tr>
<td>LOG</td>
<td>TRIMN (RTRIM)</td>
</tr>
<tr>
<td>LOG10</td>
<td>UPCASE (UCASE)</td>
</tr>
<tr>
<td>LOWCASE (LCASE)</td>
<td>WEEKDAY (DAYOFWEEK)</td>
</tr>
<tr>
<td>MAX</td>
<td>YEAR</td>
</tr>
</tbody>
</table>
Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to DB2. Due to incompatibility in date and time functions between DB2 and SAS, DB2 might not process them correctly. Check your results to determine whether these functions are working as expected.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>DBMS Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESS (REPLACE)</td>
<td>SOUNDEX</td>
</tr>
<tr>
<td>DATE (CURDATE)</td>
<td>TIME (CURTIME)</td>
</tr>
<tr>
<td>DATEPART (DATE)</td>
<td>TIMEPART</td>
</tr>
<tr>
<td>DATETIME (NOW)</td>
<td>TODAY (CURDATE)</td>
</tr>
</tbody>
</table>

Passing Joins to DB2 under UNIX and PC Hosts

For a multiple-libref join to pass to DB2, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- update isolation level (UPDATE_ISOLATION_LEVEL=, if specified)
- read_isolation level (READ_ISOLATION_LEVEL=, if specified)
- qualifier (QUALIFIER=)
- data source (DATASRC=)
- prompt (PROMPT= must not be specified)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for DB2 under UNIX and PC Hosts

Overview

Bulk loading is the fastest way to insert large numbers of rows into a DB2 table. Using this facility instead of regular SQL insert statements, you can insert rows two to ten times more rapidly. DB2 bulk-load examples and tips for maximizing performance on page 523 are available. You must specify BULKLOAD=YES to use the bulk-load facility.

SAS/ACCESS Interface to DB2 under UNIX and PC Hosts offers LOAD, IMPORT, and CLI LOAD bulk-loading methods. The BL_REMOTE_FILE= and BL_METHOD= data set options determine which method to use.

For more information about the differences between IMPORT, LOAD, and CLI LOAD, see the DB2 Data Movement Utilities Guide and Reference.
Using the LOAD Method

To use the LOAD method, you must have system administrator, database administrator, or load authority on the database and the INSERT privilege on the table to be loaded.

This method also requires that client and server machines can read and write files to a common location such as a mapped network drive or an NFS directory. To use this method, specify the `BL_REMOTE_FILE=` option.

*Note:* Because SAS/ACCESS Interface to DB2 uses the PC/IXF file format to transfer data to the DB2 LOAD utility, you cannot use this method to load data into partitioned databases.

Data Set Options with Bulk Loading

Here are the bulk-load options available with the LOAD method. For details about these options, see Data Set Options for Relational Databases on page 241.

- `BL_CODEPAGE=`
- `BL_DATAFILE=`
- `BL_DELETE_DATAFILE=`
- `BL_LOG=`: The log file contains a summary of load information and error descriptions. On most platforms, the default filename is `BL_<table>_<unique-ID>`.log, where `table` specifies the table name and `unique-ID` specifies a number used to prevent collisions in the event of two or more simultaneous instances of bulk loading of a particular table. The SAS/ACCESS engine generates this number.
- `BL_OPTIONS=`
- `BL_REMOTE_FILE=`
- `BL_SERVER_DATAFILE=`
- `BL_WARNING_COUNT=`

Using the IMPORT Method

The IMPORT method does not offer the same level of performance as the LOAD method. However, it is available to all users with INSERT privileges for the tables to be loaded. The IMPORT method does not require that the server and client have a common location in order to access the data file. If you do not specify `BL_REMOTE_FILE=`, the IMPORT method is automatically used.

Here are the bulk-load options available with the IMPORT method. For detailed information about these options, see “Overview” on page 241.

- `BL_CODEPAGE=`
- `BL_DATAFILE=`
- `BL_DELETE_DATAFILE=`
- `BL_LOG=`
- `BL_OPTIONS=`
Using the CLI LOAD Method

The CLI LOAD method is an interface to the standard DB2 LOAD utility, which gives the added performance of using LOAD but without setting additional options for bulk loading. This method requires the same privileges as the LOAD method, and is available only in DB2 Version 7 FixPak 4 and later clients and servers. If your client and server can support the CLI LOAD method, you can generally see the best performance by using it. The CLI LOAD method can also be used to load data into a partitioned DB2 database for client and database nodes that are DB2 Version 8.1 or later. To use this method, specify BL_METHOD=CLILOAD as a data set option. Here are the bulk-load options that are available with the CLI LOAD method:

- BL_ALLOW_READ_ACCESS=
- BL.Allow_WRITE_ACCESS=
- BL_COPY_LOCATION=
- BL_CPU_PARALLELISM=
- BL_DATA_BUFFER_SIZE=
- BL_DISK_PARALLELISM=
- BL_EXCEPTION
- BL_INDEXING_MODE=
- BL_LOAD_REPLACE=
- BL_LOG=
- BL_METHOD=
- BL_OPTIONS=
- BL_RECOVERABLE=
- BL_REMOTE_FILE=

Capturing Bulk-Load Statistics into Macro Variables

These bulk-load macro variables capture how many rows are loaded, skipped, rejected, committed, and deleted before writing this information to the SAS log.

- SYSBL_ROWSCOMMITTED
- SYSBL_ROWSDELETED
- SYSBL_ROWSLOADED
- SYSBL_ROWSREJECTED
- SYSBL_ROWSSKIPPED

Maximizing Load Performance for DB2 under UNIX and PC Hosts

These tips can help you optimize LOAD performance when you are using the DB2 bulk-load facility.
• Specifying BL_REMOTE_FILE= causes the loader to use the DB2 LOAD utility. This is much faster than the IMPORT utility, but it requires database administrator authority.

• Performance might suffer if your setting for DBCOMMIT= is too low. Increase the default (10000 when BULKLOAD=YES) for improved performance.

• Increasing the DB2 tuning parameters, such as Utility Heap and input-output characteristics, improves performance. These parameters are controlled by your database or server administrator.

• When using the IMPORT utility, specify BL_OPTIONS="COMPOUND=x"—where x is a number between 1 and 7 on Windows, and between 1 and 100 on UNIX. This causes the IMPORT utility to insert multiple rows for each execute instead of one row per execute.

• When using the LOAD utility on a multi-processor or multi-node DB2 server, specify BL_OPTIONS="ANYORDER" to improve performance. This might cause DB2 log entries to be out of order because it lets DB2 insert rows in a different order from how they appear in the loader data file.

Examples

This example shows how to use a SAS data set, SASFLT.FLT98, to create and load a large DB2 table, FLIGHTS98. Because the code specifies BULKLOAD=YES and BL_REMOTE_FILE= is omitted, this load uses the DB2 IMPORT command.

```
libname sasflt 'SAS-library';
libname db2_air db2 user=myuser using=mypwd
database='db2_flt' schema=statsdiv;
proc sql;
create table db2_air.flights98
(bulkload=YES bl_options='compound=7 norowwarnings')
as select * from sasflt.flt98;
quit;
```

The BL_OPTIONS= option passes DB2 file type modifiers to DB2. The norowwarnings modifier indicates that all row warnings about rejected rows are to be suppressed.

This example shows how to append the SAS data set, SASFLT.FLT98 to a pre-existing DB2 table, ALLFLIGHTS. Because the code specifies BULKLOAD=YES and BL_REMOTE_FILE=, this load uses the DB2 LOAD command.

```
proc append base=db2_air.allflights
(bulkload=YES
 BL_REMOTE_FILE='/tmp/tmpflt'
 BL_LOG='/tmp/fltdata.log'
 BL_DATAFILE='/nfs/server/tmp/fltdata.ixf'
 BL_SERVER_DATAFILE='/tmp/fltdata.ixf')
data=sasflt.flt98;
run;
```

Here, BL_REMOTE_FILE= and BL_SERVER_DATAFILE= are paths relative to the server. BL_LOG= and BL_DATAFILE= are paths relative to the client.

This example shows how to use the SAS data set SASFLT.ALLFLIGHTS to create and load a large DB2 table, ALLFLIGHTS. Because the code specifies BULKLOAD=YES
and BL_METHOD=CLILOAD, this operation uses the DB2 CLI LOAD interface to the
LOAD command.

data db2_air.allflights(BULKLOAD=YES BL_METHOD=CLILOAD);
set sasflt.allflights;
run;

Locking in the DB2 under UNIX and PC Hosts Interface

These LIBNAME and data set options let you control how the DB2 under UNIX and PC
Hosts interface handles locking. For general information about an option, see
"LIBNAME Options for Relational Databases" on page 101. For additional information,
see your DB2 documentation.

READ_LOCK_TYPE= ROW | TABLE
UPDATE_LOCK_TYPE= ROW | TABLE
READ_ISOLATION_LEVEL= RR | RS | CS | UR

The DB2 database manager supports the RR, RS, CS, and UR isolation levels that
are defined in the following table. Regardless of the isolation level, the database
manager places exclusive locks on every row that is inserted, updated, or deleted. All
isolation levels therefore ensure that only this application process can change any
given row during a unit of work: No other application process can change any rows
until the unit of work is complete.

Table 16.3 Isolation Levels for DB2 under UNIX and PC Hosts

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR (Repeatable Read)</td>
<td>no dirty Reads, no nonrepeatable Reads, no phantom Reads</td>
</tr>
<tr>
<td>RS (Read Stability)</td>
<td>no dirty Reads, no nonrepeatable Reads; does allow phantom Reads</td>
</tr>
<tr>
<td>CS (Cursor Stability)</td>
<td>no dirty Reads; does allow nonrepeatable Reads and phantom Reads</td>
</tr>
<tr>
<td>UR (Uncommitted Read)</td>
<td>allows dirty Reads, nonrepeatable Reads, and phantom Reads</td>
</tr>
</tbody>
</table>

Here is how the terms in the table are defined.

Dirty reads
A transaction that exhibits this phenomenon has very minimal isolation from
concurrent transactions. In fact, it can see changes that those concurrent
transactions made even before they commit them.

For example, suppose that transaction T1 performs an update on a row,
transaction T2 then retrieves that row, and transaction T1 then terminates with
rollback. Transaction T2 has then seen a row that no longer exists.

Nonrepeatable reads
If a transaction exhibits this phenomenon, then the system read a row once and
was unable to read the row again later in the same transaction. This might occur
if a concurrent transaction changed or even deleted the row. Therefore, the read is not (necessarily) repeatable.

For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.

**Phantom reads**

When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist (a “phantom”).

**UPDATE_ISOLATION_LEVEL= CS | RS | RR**

The DB2 database manager supports the CS, RS, and RR isolation levels defined in the preceding table. Uncommitted reads are not allowed with this option.

---

**Naming Conventions for DB2 under UNIX and PC Hosts**

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) DB2 is not case sensitive, and all names default to uppercase.

DB2 objects include tables, views, columns, and indexes. They follow these naming conventions.

- A name can begin with a letter or one of these symbols: dollar sign ($), number or pound sign (#), or at symbol (@).
- A table name must be from 1 to 128 characters long. A column name must be from 1 to 32 characters long.
- A name can contain the letters A to Z, any valid letter with a diacritic, numbers from 0 to 9, underscore (_), dollar sign ($), number or pound sign (#), or at symbol (@).
- Names are not case sensitive. For example, the table names CUSTOMER and Customer are the same, but object names are converted to uppercase when they are entered. If a name is enclosed in quotation marks, the name is case sensitive.
- A name cannot be a DB2 reserved word or an SQL-reserved word, such as WHERE or VIEW.
- A name cannot be the same as another DB2 object that has the same type.

Schema and database names have similar conventions, except that they are each limited to 30 and 8 characters respectively. For more information, see your DB2 SQL reference documentation.
Data Types for DB2 under UNIX and PC Hosts

Overview

Every column in a table has a name and a data type. The data type tells DB2 how much physical storage to set aside for the column and the form in which the data is stored. DB2 uses IBM SQL data types. This section includes information about DB2 data types, null and default values, and data conversions.

For more information about DB2 data types and to determine which data types are available for your version of DB2, see your DB2 SQL reference documentation.

Supported Data Types for DB2 under UNIX and PC Hosts

Here are the data types that are supported for DB2 under UNIX and PC Hosts:

- **Character data:**
  - CHAR\((n)\)
  - CLOB (character large object)
  - GRAPHIC\((n)\)
  - LONG VARGRAPHIC
  - VARCHAR\((n)\)
  - VARGRAPHIC\((n)\)
  - LONG VARCHAR

- **Numeric data:**
  - BIGINT
  - INTEGER
  - DECIMAL | DEC | NUMERIC | NUM
  - SMALLINT
  - FLOAT | DOUBLE | DOUBLE PRECISION

  *Note:* When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

- **Date and time data:**
  - DATE
  - TIMESTAMP
  - TIME

  *Note:* SQL date and time data types are collectively called datetime values. Be aware that columns of these data types can contain data values that are out of range for SAS.

- **Binary data:** BLOB (binary large object)

For more information about DB2 data types, see your DB2 documentation.
**DB2 Null and Default Values**

DB2 has a special value called NULL. A DB2 NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DB2 NULL value, it interprets it as a SAS missing value.

You can define a column in a DB2 table so that it requires data. To do this in SQL, you specify a column as NOT NULL. NOT NULL tells SQL to only allow a row to be added to a table if there is a value for the field. For example, NOT NULL assigned to the field CUSTOMER in the table SASDEMO.CUSTOMER does not allow a row to be added unless there is a value for CUSTOMER. When creating a DB2 table with SAS/ACCESS, you can use the `DBNULL=` data set option to indicate whether NULL is a valid value for specified columns.

DB2 columns can also be defined as NOT NULL WITH DEFAULT. For more information about using the NOT NULL WITH DEFAULT value, see your DB2 SQL reference documentation.

Once you know whether a DB2 column enables NULLs or the host system supplies a default value for a column that is defined as NOT NULL WITH DEFAULT, you can write selection criteria and enter values to update a table. Unless a column is defined as NOT NULL or NOT NULL WITH DEFAULT, it allows NULL values.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how the DBMS handles SAS missing character values, use the `NULLCHAR=` and `NULLCHARVAL=` data set options.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to DB2 assigns to SAS variables when using the `LIBNAME statement` to read from a DB2 table. These default formats are based on DB2 column attributes.

**Table 16.4 LIBNAME Statement: Default SAS Formats for DB2 Data Types**

<table>
<thead>
<tr>
<th>DB2 Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>character</td>
<td>$HEXw:</td>
</tr>
<tr>
<td>CLOB</td>
<td>character</td>
<td>$w:</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>character</td>
<td>$w:</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRAPHIC(n)</td>
<td>character</td>
<td>$w:</td>
</tr>
<tr>
<td>VARGRAPHIC(n)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONG VARGRAPHIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>numeric</td>
<td>11.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 Data Type</td>
<td>SAS Data Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.</td>
</tr>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>FLOAT</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME8.</td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME w.d</td>
</tr>
</tbody>
</table>

*  

n in DB2 character and graphic data types is equivalent to w in SAS formats.

This table shows the default DB2 data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

**Table 16.5 LIBNAME Statement: Default DB2 Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>DB2 Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.d</td>
<td>DECIMAL (m,n)**</td>
</tr>
<tr>
<td>other numerics</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>$w.</td>
<td>VARCHAR(n) (n&lt;=4000)</td>
</tr>
<tr>
<td></td>
<td>LONG VARCHAR(n) (n&gt;4000)</td>
</tr>
<tr>
<td>datetime formats</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>TIME</td>
</tr>
</tbody>
</table>

*  

n in DB2 data types is equivalent to w in SAS formats.

**  

m and n in DB2 numeric data types are equivalent to w and d in SAS formats.

**DBLOAD Procedure Data Conversions**

This table shows the default DB2 data types that SAS/ACCESS assigns to SAS variable formats when you use the DBLOAD procedure.
Table 16.6  PROC DBLOAD: Default DB2 Data Types for SAS Variable Formats

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>DB2 Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.</td>
<td>CHAR($w)</td>
</tr>
<tr>
<td>w.</td>
<td>DECIMAL($w)</td>
</tr>
<tr>
<td>w.d</td>
<td>DECIMAL($w,d)</td>
</tr>
<tr>
<td>IBw.d, PIBw.d</td>
<td>INTEGER</td>
</tr>
<tr>
<td>all other numerics***</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>datetimew.d</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>datew.</td>
<td>DATE</td>
</tr>
<tr>
<td>time.†</td>
<td>TIME</td>
</tr>
</tbody>
</table>

* $n$ in DB2 character and graphic data types is equivalent to $w$ in SAS formats.
** $p$ and $s$ in DB2 numeric data types are equivalent to $w$ and $d$ in SAS formats.
*** Includes all SAS numeric formats, such as BINARY8 and E10.0.
† Includes all SAS time formats, such as TODw,d and HHMMw,d.
## Chapter 17
### SAS/ACCESS Interface to DB2 for z/OS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to DB2 under z/OS</td>
<td>533</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for DB2 under z/OS</td>
<td>533</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>533</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>533</td>
</tr>
<tr>
<td>DB2 under z/OS LIBNAME Statement Example</td>
<td>536</td>
</tr>
<tr>
<td>Data Set Options for DB2 under z/OS</td>
<td>536</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for DB2 under z/OS</td>
<td>539</td>
</tr>
<tr>
<td><strong>Key Information</strong></td>
<td>539</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>540</td>
</tr>
<tr>
<td>Autopartitioning Scheme for DB2 under z/OS</td>
<td>540</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>540</td>
</tr>
<tr>
<td><strong>Autopartitioning Restrictions</strong></td>
<td>541</td>
</tr>
<tr>
<td>Column Selection for MOD Partitioning</td>
<td>541</td>
</tr>
<tr>
<td>How WHERE Clauses Restrict Autopartitioning</td>
<td>541</td>
</tr>
<tr>
<td>Using DBSLICE=</td>
<td>542</td>
</tr>
<tr>
<td>Using DBSLICEPARM=</td>
<td>542</td>
</tr>
<tr>
<td>Temporary Table Support for DB2 under z/OS</td>
<td>542</td>
</tr>
<tr>
<td>Calling Stored Procedures in DB2 under z/OS</td>
<td>542</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>542</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>542</td>
</tr>
<tr>
<td>ACCESS Procedure Specifics for DB2 under z/OS</td>
<td>544</td>
</tr>
<tr>
<td><strong>Key Information</strong></td>
<td>544</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>545</td>
</tr>
<tr>
<td>DBLOAD Procedure Specifics for DB2 under z/OS</td>
<td>546</td>
</tr>
<tr>
<td><strong>Key Information</strong></td>
<td>546</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>547</td>
</tr>
<tr>
<td>The DB2EXT Procedure</td>
<td>547</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>547</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td>547</td>
</tr>
<tr>
<td>PROC DB2EXT Statement Options</td>
<td>548</td>
</tr>
<tr>
<td>FMT Statement</td>
<td>548</td>
</tr>
<tr>
<td>RENAME Statement</td>
<td>548</td>
</tr>
<tr>
<td>SELECT Statement</td>
<td>549</td>
</tr>
<tr>
<td>EXIT Statement</td>
<td>549</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>549</td>
</tr>
</tbody>
</table>
Introduction to SAS/ACCESS Interface to DB2 under z/OS

For available SAS/ACCESS features, see DB2 under z/OS supported features on page 79. For more information about DB2 under z/OS, see your DB2 under z/OS documentation.

LIBNAME Statement Specifics for DB2 under z/OS

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to DB2 under z/OS supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing DB2 under z/OS interface.

```sas
LIBNAME libref db2 <connection-options> <LIBNAME-options>;
```

Arguments

- **libref**
  - specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

- **db2**
  - specifies the SAS/ACCESS engine name for the DB2 under z/OS interface.

- **connection-options**
  - provides connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. Here is how these options are defined.

  - `USER=<''DB2-user-name''>`
    - lets you connect to a DB2 database with a user ID that is different from the default ID. The value for this option cannot exceed 8 characters. `USER=` is optional. If you specify `USER=`, you must also specify `PASSWORD=`. If `USER=` is omitted, your default user ID for your operating environment is used. If you do not specify the `SCHEMA=` or `AUTHID=` LIBNAME option, the value of the `USER=` option (if present) is used as the default schema. Authentication options are not checked when the LIBNAME is issued. They are checked only when a statement involving the libref is run. However, for explicit pass-through, credentials are checked immediately.

  - `PASSWORD=<''DB2-password''>`
    - specifies the DB2 password that is associated with your DB2 user ID. `PASSWORD=` is optional. If you specify `USER=`, you must specify `PASSWORD=`.

  - `LOCATION=location`
    - maps to the location in the SYSIBM.LOCATIONS catalog in the communication database. In SAS/ACCESS Interface to DB2 under z/OS, the location is converted to the first level of a three-level table name: `location.authid.table`. DB2
Distributed Data Facility (DDF) Communication Database (CDB) makes the connection implicitly to the remote DB2 subsystem when DB2 receives a three-level name in an SQL statement.

LOCATION= is optional. If you omit it, SAS accesses the data from the local DB2 database unless you have specified a value for the SERVER= option. This option is not validated until you access a DB2 table. If you specify LOCATION=, you must also specify the AUTHID= on page 106 option.

SSID=DB2-subsystem-id
specifies the DB2 subsystem ID to connect to at connection time. SSID= is optional. If you omit it, SAS connects to the DB2 subsystem that is specified in the DB2SSID= SAS system option. The DB2 subsystem ID is limited to four characters. For more information, see “Settings” on page 560.

SERVER=DRDA-server
specifies the DRDA server to which you want to connect. SERVER= lets you access DRDA resources stored at remote locations. Check with your system administrator for system names. You can connect to only one server per LIBNAME statement.

SERVER= is optional. If you omit it, you access tables from your local DB2 database unless you have specified a value for the LOCATION= LIBNAME option on page 171.

Default: none.

<table>
<thead>
<tr>
<th>For information about</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>accessing a database server on Linux, UNIX, or Windows using a libref</td>
<td>REMOTE_DBTYPE= LIBNAME option</td>
</tr>
<tr>
<td>setting up DB2 z/OS so that SAS can connect to the DRDA server when the SERVER= option is used</td>
<td>installation instructions for this interface</td>
</tr>
<tr>
<td>configuring SAS to use the SERVER= option</td>
<td></td>
</tr>
</tbody>
</table>

**LIBNAME-options**
defines how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to DB2 under z/OS, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

**Table 17.1  SAS/ACCESS LIBNAME Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>ALLOWED_SQLCODES=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAINE=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>AUTHID=</td>
<td>your user ID</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DLBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DLBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DEGREE=</td>
<td>ANY</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IN=</td>
<td>none</td>
</tr>
<tr>
<td>LOCATION=</td>
<td>none</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>NONE</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>NO</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>NO</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>1</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>DB2 z/OS determines the isolation level</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>none</td>
</tr>
</tbody>
</table>
Option | Default Value
---|---
REMOTE_DBTYPE= | ZOS
REREAD_EXPOSURE= | NO
SCHEMA= | your user ID
SPOOL= | YES
SQL_FUNCTIONS= | none
SQL_FUNCTIONS_COPY= | none
UPDATE_ISOLATION_LEVEL= | DB2 z/OS determines the isolation level
UPDATE_LOCK_TYPE= | none
UTILCONN_TRANSIENT= | YES

**DB2 under z/OS LIBNAME Statement Example**

In this example, the libref MYLIB uses the DB2 under z/OS interface to connect to the DB2 database that the SSID= option specifies, with a connection to the *testserver* remote server.

```sql
libname mylib db2 ssid=db2
   authid=myusr1 server=mysrv1;
proc print data=mylib.staff;
  where state='CA';
run;
```

**Data Set Options for DB2 under z/OS**

All SAS/ACCESS data set options in this table are supported for SAS/ACCESS Interface to DB2 under z/OS. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHID=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>BL_DBCURSOR=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DB2DATAACLAS=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>BL_DB2DEVT_PERM=</td>
<td>SYSDA</td>
</tr>
<tr>
<td>BL_DB2DEVT_TEMP=</td>
<td>SYSDA</td>
</tr>
<tr>
<td>BL_DB2DISC=</td>
<td>a generated data set name</td>
</tr>
<tr>
<td>BL_DB2ERR=</td>
<td>a generated data set name</td>
</tr>
<tr>
<td>BL_DB2IN=</td>
<td>a generated data set name</td>
</tr>
<tr>
<td>BL_DB2LDCT1=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DB2LDCT2=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DB2LDCT3=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DB2LDEXT=</td>
<td>GENRUN</td>
</tr>
<tr>
<td>BL_DB2MAP=</td>
<td>a generated data set name</td>
</tr>
<tr>
<td>BL_DB2MGMTCLAS=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DB2PRINT=</td>
<td>a generated data set name</td>
</tr>
<tr>
<td>BL_DB2PRNLOG=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_DB2REC=</td>
<td>a generated data set name</td>
</tr>
<tr>
<td>BL_DB2RECSP=</td>
<td>10</td>
</tr>
<tr>
<td>BL_DB2STRT=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_DB2SPC_PERM=</td>
<td>10</td>
</tr>
<tr>
<td>BL_DB2SPC_TEMP=</td>
<td>10</td>
</tr>
<tr>
<td>BL_DB2STORCLAS=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DB2TBLXST=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_DB2UNITCOUNT=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DB2UTID=</td>
<td>user ID and second level DSN qualifier</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>BUSINESS_DATATYPE=</td>
<td>TIMESTAMP(6)</td>
</tr>
<tr>
<td>BUSINESS_TIMEFRAME=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDition=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DLBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBASTYPE=</td>
<td>see “Data Types for DB2 under z/OS”</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARAM=</td>
<td>THREADED_APPS,2</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>none</td>
</tr>
<tr>
<td>DEGREE=</td>
<td>ANY</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>IN=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>LOCATION=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>OVERLAPS=</td>
<td>column names separated by commas</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
</tbody>
</table>
SQL Pass-Through Facility Specifics for DB2 under z/OS

Key Information

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the DB2 under z/OS interface:

- The dbms-name is DB2.
- The CONNECT statement is optional.
- The interface supports connections to multiple databases.
- For DB2 V8 or higher, the maximum length of an SQL statement is 1 megabyte.
- Although you can specify in the CONNECT statement any database-connection-arguments from the LIBNAME statement, only the following connection arguments are recognized:
  - DEGREE=
  - LOCATION=
  - PASSWORD=
  - READBUFF=
  - SERVER=
  - SSID=
Examples

This example connects to DB2 and sends it two EXECUTE statements to process.

```sql
proc sql;
    connect to db2 (ssid=db2);
    execute (create view testid.whotookorders as
        select ordernum, takenby, firstname, lastname, phone
        from testid.orders, testid.employees
        where testid.orders.takenby=
            testid.employees.empid)
        by db2;
    execute (grant select on testid.whotookorders
        to myusr1) by db2;
    disconnect from db2;
quit;
```

This next example omits the optional CONNECT statement, uses the default DB2SSID= setting, and performs a query (shown in highlighting) on the Testid.Customers table.

```sql
proc sql;
    select * from connection to db2
        (select * from testid.customers where customer like '1%');
    disconnect from db2;
quit;
```

This example creates the Vlib.StockOrd SQL view that is based on the Testid.Orders table. Testid.Orders is an SQL/DS table that is accessed through DRDA.

```sql
libname vlib 'SAS-library'

proc sql;
    connect to db2 (server=testserver);
    create view vlib.stockord as
        select * from connection to db2
            (select ordernum, stocknum, shipto, dateorderd
            from testid.orders);
    disconnect from db2;
quit;
```

Autopartitioning Scheme for DB2 under z/OS

Overview

Autopartitioning for SAS/ACCESS Interface to DB2 under z/OS is a modulo (MOD) method. Threaded Reads for DB2 under z/OS involve a trade-off. A threaded Read with even distribution of rows across the threads substantially reduces elapsed time for your SAS step. So your job completes in less time. This is positive for job turnaround time, particularly if your job needs to complete within a constrained period of time. However, threaded Reads always increase the CPU time of your SAS job and the workload on DB2. If increasing CPU consumption or increasing DB2 workload for your job are
unacceptable, you can turn threaded Reads off by specifying DBSLICEPARM=NONE. To turn off threaded Reads for all SAS jobs, set DBSLICEPARM=NONE in the SAS restricted options table.

For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

**Autopartitioning Restrictions**

SAS/ACCESS Interface to DB2 under z/OS places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here are the column types that you can partition.

- INTEGER
- SMALLINT
- DECIMAL

You must confine eligible DECIMAL columns to an integer range—specifically, DECIMAL columns with precision that is less than 10. For example, DECIMAL(5,0) and DECIMAL(9,2) are eligible.

**Column Selection for MOD Partitioning**

If multiple columns are eligible for partitioning, the engine queries the DB2 system tables for information about identity columns and simple indexes. Based on the information about the identity columns, simple indexes, column types, and column nullability, the partitioning column is selected in order by priority.

1. Identity column
2. Unique simple index: SHORT or INT, integral DECIMAL, and then nonintegral DECIMAL
3. Nonunique simple index: SHORT or INT (NOT NULL), integral DECIMAL (NOT NULL), and then nonintegral DECIMAL (NOT NULL)
4. Nonunique simple index: SHORT or INT (nullable), integral DECIMAL (nullable), and then nonintegral DECIMAL (nullable)
5. SHORT or INT (NOT NULL), integral DECIMAL (NOT NULL), and then nonintegral DECIMAL (NOT NULL)
6. SHORT or INT (nullable), integral DECIMAL (nullable), and then nonintegral DECIMAL (nullable)

If a nullable column is selected for autopartitioning, the SQL statement `OR<column-name>IS NULL` is appended at the end of the SQL code that is generated for one read thread. This ensures that any possible NULL values are returned in the result set.

**How WHERE Clauses Restrict Autopartitioning**

Autopartitioning does not select a column to be the partitioning column if it appears in a SAS WHERE clause. For example, the following DATA step cannot use a threaded Read to retrieve the data because all numeric columns in the table (see the table definition in “Using DBSLICE=” on page 542) are in the WHERE clause.

```sql
data work.locemp;
  set trlib.MYEMPS;
```
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;

Using DBSLICEPARM=
SAS/ACCESS Interface to DB2 under z/OS defaults to two threads when you use autopartitioning.

Using DBSLICE=
You can achieve the best possible performance when using threaded Reads by specifying the DBSLICE= data set option for DB2 in your SAS operation.

Temporary Table Support for DB2 under z/OS
SAS/ACCESS Interface to DB2 under z/OS supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

Calling Stored Procedures in DB2 under z/OS
Overview
A stored procedure is one or more SQL statements or supported third-generation languages (3GLs, such as C) statements that are compiled into a single procedure that exists in DB2. Stored procedures might contain static (hardcoded) SQL statements. Static SQL is optimized better for some DBMS operations. In a carefully managed DBMS environment, programmers and database administrators can know the exact SQL to execute.

SAS usually generates SQL dynamically. However, the database administrator can encode static SQL in a stored procedure and therefore restrict SAS users to a tightly controlled interface. When you use a stored procedure call, you must specify a schema.

SAS/ACCESS support for stored procedure includes passing input parameters, retrieving output parameters into SAS macro variables, and retrieving the result set into a SAS table. Although DB2 stored procedures can return multiple result sets, SAS/ACCESS Interface to DB2 under z/OS can retrieve only a single result set.

You can call stored procedures only from PROC SQL.

Examples

Example 1: Specify a Basic Call
Use CALL statement syntax to call a stored procedure.

call "schema".stored_proc
The simplest way to call a stored procedure is to use the EXECUTE statement in PROC SQL. In this example, you execute STORED_PROC by using a CALL statement. SAS does not capture the result set.

```sql
proc sql;
  connect to db2;
  execute (call "schema".stored_proc);
quit;
```

**Example 2: Specify One Input Parameter That Returns a Result Set**

You can also return the result set to a SAS table. In this example, STORED_PROC is executed using a CALL statement. The result is returned to a SAS table, SasResults.

```sql
proc sql;
  connect to db2;
  create table sasresults as select * from connection
to db2 (call "schema".stored_proc);
quit;
```

**Example 3: Specify Three Output Parameters**

The CALL statement syntax supports passing of parameters. You can specify such input parameters as numeric constants, character constants, or a null value. You can also pass input parameters by using SAS macro variable references. To capture the value of an output parameter, a SAS macro variable reference is required. This example uses a constant (1), an input/output parameter (:INOUT), and an output parameters (:OUT). Not only is the result set returned to the SAS results table, the SAS macro variables INOUT and OUT capture the parameter outputs.

```sql
proc sql;
  connect to db2;
  %let INOUT=2;
  create table sasresults as select * from connection to db2
      (call "schema".stored_proc (1,:INOUT,:OUT));
quit;
```

**Example 4: Pass a NULL Parameter**

In these calls, NULL is passed as the parameter to the DB2 stored procedure.

- null string literals in the call
  ```sql
call proc('');
call proc(""
```

- literal period or literal NULL in the call
  ```sql
call proc(.)
call proc(NULL)
```

- SAS macro variable set to NULL string
  ```sql
%let charparm=;
call proc(:charparm)
```

- SAS macro variable set to period (SAS numeric value is missing)
  ```sql
%let numparm=.;
call proc(:numparm)
```

Only the literal period and the literal NULL work generically for both DB2 character parameters and DB2 numeric parameters. For example, a DB2 numeric parameter would
reject "" and %let numparm=.; would not pass a DB2 NULL for a DB2 character parameter. As a literal, a period passes NULL for both numeric and character parameters. However, when it is in a SAS macro variable, it constitutes a NULL only for a DB2 numeric parameter.

You cannot pass NULL parameters by omitting the argument. For example, you cannot use this call to pass three NULL parameters.

```sas
call proc(,,)
```

You could use this call instead.

```sas
call proc(NULL,NULL,NULL)
```

**Example 5: Specify a Schema**

Use standard CALL statement syntax to execute a stored procedure that exists in another schema, as shown in this example.

```sas
proc sql;
   connect to db2;
   execute (call otherschema.stored_proc);
quit;
```

If the schema is in mixed case or lowercase, enclose the schema name in double quotation marks.

```sas
proc sql;
   connect to db2;
   execute (call "lowschema".stored_proc);
quit;
```

**Example 6: Execute Remote Stored Procedures**

If the stored procedure exists on a different DB2 instance, specify it with a valid three-part name.

```sas
proc sql;
   connect to db2;
   create table sasresults as select * from connection to db2
      (call otherdb2.procschema.prod5 (1, NULL));
quit;
```

---

**ACCESS Procedure Specifics for DB2 under z/OS**

**Key Information**

For general information about this feature, see the ACCESS procedure on page 955.

SAS/ACCESS Interface to DB2 under z/OS supports all ACCESS procedure statements in interactive line, noninteractive, and batch modes.

Here are the ACCESS procedure specifics for the DB2 under z/OS interface.

- The DBMS= value is db2.
- The database-connection-arguments are SSID=, SERVER=, and LOCATION=.
LOCATION=location
lets you further qualify where a table is located. In the DB2 z/OS engine, the
location is converted to the first level of a three-level table name:
Location.Authid.Table. The connection to the remote DB2 subsystem is done
implicitly by DB2 when DB2 receives a three-level table name in an SQL
statement.

LOCATION= is optional. If you omit it, SAS accesses the data from the local
DB2 database.

• Here is the TABLE= statement:

  TABLE= <authorization-id.>table-name
  identifies the DB2 table or DB2 view that you want to use to create an access
descriptor. The table-name is limited to 18 characters. The TABLE= statement is
required.

  The authorization-id is a user ID or group ID that is associated with the DB2
table. The authorization ID is limited to eight characters. If you omit the
authorization ID, DB2 uses your TSO (or z/OS) user ID. In batch mode,
however, you must specify an authorization ID. Otherwise, an error message is
generated.

Examples

This example creates an access descriptor and a view descriptor that are based on DB2
data.

options linesize=80;
libname adlib 'SAS-library';
libname vlib 'SAS-library';

proc access dbms=db2;
  /* create access descriptor */
  create adlib.customr.access;
  table=testid.customers;
  ssid=db2;
  assign=yes;
  rename customer=custnum;
  format firstorder date7.;
  list all;

  /* create vlib.usacust view */
  create vlib.usacust.view;
  select customer state zipcode name
       firstorder;
  subset where customer like '1%';
run;

This next example uses the SERVER= statement to access the SQL/DS table
Testid.Orders from a remote location. Access and view descriptors are then created
based on the table.

libname adlib 'SAS-library';
libname vlib 'SAS-library';

proc access dbms=db2;
create adlib.customr.access;
table=testid.orders;
server=testserver;
assign=yes;
list all;

create vlib.allord.view;
select ordernum stocknum shipto dateorderr;
subset where stocknum = 1279;
run;

DBLOAD Procedure Specifics for DB2 under z/OS

Key Information

For general information about this feature, see the DBLOAD procedure on page 985. SAS/ACCESS Interface to DB2 under z/OS supports all DBLOAD procedure statements in interactive line, noninteractive, and batch modes.

Here are the DBLOAD procedure specifics for SAS/ACCESS Interface to DB2 under z/OS.

- The DBMS= value is **DB2**.
- The database-connection-arguments are **SSID=** and **SERVER=**.
- Here is the NULLS= statement:
  
  `NULLS variable-identifier-1 =Y|N|D < . . . variable-identifier-n =Y|N|D>`
  
  lets you specify whether DB2 columns that are associated with the listed SAS variables allow NULL values. By default, all columns accept NULL values.

  The NULLS statement accepts any one of these values.

  - Y: Specifies that the column accepts NULL values. This is the default.
  - N: Specifies that the column does not accept NULL values.
  - D: Specifies that the column is defined as NOT NULL WITH DEFAULT.

  For information about NULL values that is specific to DB2, see “DB2 Null and Default Values” on page 569.

- Here is the TABLE= statement:
  
  `TABLE= <authorization-id.>table-name;`
  
  identifies the DB2 table or DB2 view that you want to use to create an access descriptor. The `table-name` is limited to 18 characters. The TABLE= statement is required.

  The `authorization-id` is a user ID or group ID that is associated with the DB2 table. The authorization ID is limited to eight characters. If you omit the authorization ID, DB2 uses your TSO (or z/OS) user ID. However, in batch mode, you must specify an authorization ID or an error message is generated.
Examples

This example creates a new DB2 table, Testid.Invoice, from the Dlib.Invoice data file. The AmtBilled column and the fifth column in the table (AmountInUS) are renamed. You must have the appropriate privileges before you can create new DB2 tables.

```sas
libname adlib 'SAS-library';
libname dlib 'SAS-library';

proc dbload dbms=db2 data=dlib.invoice;
  ssid=db2;
  table=testid.invoice;
  accdesc=adlib.invoice;
  rename amtbilled=amountbilled
       5=amountindollars;
  nulls invoicenum=n amtbilled=n;
  load;
run;
```

For example, you can create a SAS data set, Work.Schedule, that includes the names and work hours of your employees. You can use the SERVER= command to create the DB2 table, Testid.Schedule, and load it with the schedule data on the DRDA resource, TestServer, as shown in this example.

```sas
libname adlib 'SAS-library';

proc dbload dbms=db2 data=work.schedule;
  in sample;
  server=testserver;
  accdesc=adlib.schedule;
  table=testid.schedule;
  list all;
  load;
run;
```

The DB2EXT Procedure

Overview

The DB2EXT procedure creates SAS data sets from DB2 under z/OS data. PROC DB2EXT runs interactively, noninteractively, and in batch mode. The generated data sets are not password protected. However, you can edit the saved code to add password protection.

PROC DB2EXT ensures that all SAS names that are generated from DB2 column values are unique. A numeric value is appended to the end of a duplicate name. If necessary, the procedure truncates the name when appending the numeric value.

Syntax

Here is the syntax for the DB2EXT procedure.
**PROC DB2EXT**<options>;  
  **FMT** column-number-1='SAS-format-name-1'  
  <... column-number-n='SAS-format-name-n'>;  
  **RENAME** column-number-1='SAS-name-1'  
  <... column-number-n='SAS-name-n'>;  
  **SELECT** DB2-SQL-statement;  
  **EXIT**;

**PROC DB2EXT Statement Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>**IN=**SAS-data-set</td>
<td>specifies a mapping data set that contains information such as DB2 names, SAS variable names, and formats for input to PROC DB2EXT. This option is available for use only with previously created mapping data sets. You cannot create new mapping data sets with DB2EXT.</td>
</tr>
<tr>
<td>**OUT=**SAS-data-set</td>
<td>specifies the name of the SAS data set that is created. If you omit OUT=, the data set is named &quot;work.DATAn&quot;, where n is a number that is sequentially updated. The data set is not saved when your SAS session ends. If a file with the name that you specify in the OUT= option already exists, it is overwritten. However, you receive a warning that this is going to happen.</td>
</tr>
<tr>
<td>**SSID=**subsystem-name</td>
<td>specifies the name of the DB2 subsystem that you want to access. If you omit SSID=, the subsystem name defaults to DB2. The subsystem name defaults to the subsystem that is defined in the DB2SSID= option. It defaults to DB2 only if neither the SSID= option nor the DB2SSID= option are specified.</td>
</tr>
</tbody>
</table>

**FMT Statement**

**FMT** column-number-1='SAS-format-name-1'  
  <... column-number-n='SAS-format-name-n'>;  

The FMT statement assigns a SAS output format to the DB2 column that is specified by column-number. The column-number is determined by the order in which you list the columns in your SELECT statement. If you use SELECT *, the column-number is determined by the order of the columns in the database. You must enclose the format name in single quotation marks. You can specify multiple column formats in a single FMT statement.

**RENAME Statement**

**RENAME** column-number-1='SAS-name-1'  
  <... column-number-n='SAS-name-n'>;  

The RENAME statement assigns the SAS-name to the DB2 column that is specified by column-number. The column-number is determined by the order in which you list the columns in your SELECT statement. If you use SELECT *, the column-number is determined by the order of the columns in the database.

You can rename multiple columns in a single RENAME statement.
The DB2UTIL Procedure

Overview

You can use the DB2UTIL procedure to insert, update, or delete rows in a DB2 table using data from a SAS data set. You can choose one of two methods of processing:

**SELECT Statement**

```sql
SELECT DB2-SQL-statement;
```

The `DB2-SQL-statement` defines the DB2 data that you want to include in the SAS data set. You can specify table names, column names, and data subsets in your `SELECT` statement. For example, this statement selects all columns from the Employee table and includes only employees whose salary is greater than $40,000.

```
select * from employee where salary > 40000;
```

**EXIT Statement**

```sql
EXIT;
```

The EXIT statement terminates the procedure without further processing.

**Examples**

This code creates a SAS data set named MyLib.NoFmt that includes three columns from the DB2 table EmplInfo. The RENAME statement changes the name of the third column that is listed in the `SELECT` statement (from `firstname` in the DB2 table to `fname` in the SAS data set.

```
/* specify the SAS library where the SAS data set is to be saved */
libname mylib 'userid.xxx';

proc db2ext ssid=db25 out=mylib.nofmt;
   select employee, lastname, firstname from sasdemo.emplinfo;
   rename 3=fname;
run;
```

This code uses a mapping file to specify which data to include in the SAS data set and how to format that data.

```
/* specify the SAS library where the SAS data set is to be saved */
libname mylib 'userid.xxx';

/* specify the SAS library that contains the mapping data set */
libname inlib 'userid.maps';

proc db2ext in=inlib.mapping out=mylib.mapout ssid=db25;
run;
```
creating an SQL output file or executing directly. PROC DB2UTIL runs interactively, noninteractively, or in batch mode.

Support for the DB2UTIL procedure provides compatibility with SAS 5 version of SAS/ACCESS Interface to DB2 under z/OS. It is not added to other SAS/ACCESS DBMS interfaces, and enhancement of this procedure for future releases of SAS/ACCESS are not guaranteed. It is recommended that you write new applications by using LIBNAME features.

The DB2UTIL procedure uses the data in an input SAS data set, along with your mapping specifications, to generate SQL statements that modify the DB2 table. The DB2UTIL procedure can perform these functions.

DELETE
  deletes rows from the DB2 table according to the search condition that you specify.

INSERT
  builds rows for the DB2 table from the SAS observations, according to the map that you specify, and inserts the rows.

UPDATE
  sets new column values in your DB2 table by using the SAS variable values that are indicated in your map.

When you execute the DB2UTIL procedure, you specify an input SAS data set, an output DB2 table, and how to modify the data. To generate data, you must also supply instructions for mapping the input SAS variable values to the appropriate DB2 columns.

In each execution, the procedure can generate and execute SQL statements to perform one type of modification only. However, you can also supply your own SQL statements (except the SQL SELECT statement) to perform various modifications against your DB2 tables, and the procedure executes them.

For more information about the types of modifications that are available and how to use them, see “Modifying DB2 Data” on page 552. For an example of how to use this procedure, see the PROC DB2UTIL example on page 553.

**DB2UTIL Syntax**

The PROC DB2UTIL statement calls the DB2UTIL procedure on page 549. These statements are used with PROC DB2UTIL.

```
PROC DB2UTIL <options>;
  MAPTO SAS-name-1=DB2-name-1 <…SAS-name-n=DB2-name-n>;
  RESET ALL|SAS-name|COLS;
  SQL SQL-statement;
  UPDATE;
  WHERE SQL-WHERE-clause;
  ERRLIMIT=error-limit;
  EXIT;
```

**DB2UTIL Options**

- **DATA=SAS-data-set | <libref.>SAS-data-set**
  specifies the name of the SAS data set that contains the data with which you want to update the DB2 table. DATA= is required unless you specify an SQL file with the SQLIN= option.
The DB2UTIL Procedure 551

TABLE=DB2-tablename
specifies the name of the DB2 table that you want to update. TABLE= is required unless you specify an SQL file with the SQLIN= option.

FUNCTION= D | I | U | DELETE | INSERT | UPDATE
specifies the type of modification to perform on the DB2 table by using the SAS data set as input. For a detailed description of this option, see “Modifying DB2 Data” on page 552. FUNCTION= is required unless you specify an SQL file with the SQLIN= option.

COMMIT=number
specifies the maximum number of SQL statements to execute before issuing an SQL COMMIT statement to establish a synchpoint. The default is 3.

ERROR=fileref | fileref.member
specifies an external file where error information is logged. When DB2 issues an error return code, the procedure writes all relevant information, including the SQL statement that is involved, to this external file. If you omit the ERROR= statement, the procedure writes the error information to the SAS log.

LIMIT=number
specifies the maximum number of SQL statements to issue in an execution of the procedure. The default value is 5000. If you specify LIMIT=0, no limit is set. The procedure processes the entire data set regardless of its size.

SQLIN=fileref | fileref.member
specifies an intermediate SQL output file that is created by a prior execution of PROC DB2UTIL by using the SQLOUT= option. The file that is specified by SQLIN= contains SQL statements to update a DB2 table. If you specify an SQLIN= file, the procedure reads the SQL statements and executes them in line mode. When you specify an SQLIN= file, DATA=, TABLE=, and SQLOUT= are ignored.

SQLOUT=fileref | fileref.member
specifies an external file where the generated SQL statements are to be written. This file is either a z/OS sequential data set or a member of a z/OS partitioned data set. Use this option to update or delete data. When you specify the SQLOUT= option, the procedure edits your specifications, generates the SQL statements to perform the update, and writes them to the external file for later execution. When they are input to the later run for execution, the procedure passes them to DB2.

SSID=subsystem-name
specifies the name of the DB2 subsystem that you want to access. If you omit DB2SSID=, the subsystem name defaults to DB2. For more information, see “Settings” on page 560.

DB2UTIL Statements

MAPTO Statement
MAPTO SAS-name-1=DB2-name-1<… SAS-name-n=DB2-name-n>;
The MAPTO statement maps the SAS variable name to the DB2 column name. You can specify as many values in one MAPTO statement as you want.

RESET Statement
RESET ALL | SAS-name | COLS;
Use the RESET statement to erase the editing that was done to SAS variables or DB2 columns. The RESET statement can perform one or more of these actions:
ALL
  resets all previously entered map and column names to default values for the
  procedure.

SAS-name
  resets the map entry for that SAS variable.

COLS
  resets the altered column values.

**SQL Statement**

```
SQL SQL-statement;
```

The SQL statement specifies an SQL statement that you want the procedure to execute
dynamically. The procedure rejects SQL SELECT statements.

**UPDATE Statement**

```
UPDATE;
```

The UPDATE statement causes the table to be updated by using the mapping
specifications that you supply. If you do not specify an input or output mapping data set
or an SQL output file, the table is updated by default.

If you have specified an output mapping data set in the SQLOUT= option, PROC
DB2UTIL creates the mapping data set and ends the procedure. However, if you specify
UPDATE, the procedure creates the mapping data set and updates the DB2 table.

**WHERE Statement**

```
WHERE SQL-WHERE-clause;
```

The WHERE statement specifies the SQL WHERE clause that you want to use to update
the DB2 table. This statement is combined with the SQL statement generated from your
mapping specifications. Any SAS variable names in the WHERE clause are substituted
at that time, as shown in this example.

```
where db2col = %sasvar;
```

**ERRLIMIT Statement**

```
ERRLIMIT=error-limit;
```

The ERRLIMIT statement specifies the number of DB2 errors that are permitted before
the procedure terminates.

**EXIT Statement**

```
EXIT;
```

The EXIT statement exits from the procedure without further processing. No output data
is written, and no SQL statements are issued.

---

**Modifying DB2 Data**

**Overview**

The DB2UTIL procedure generates SQL statements by using data from an input SAS
data set. However, the SAS data set plays a different role for each type of modification
that is available through PROC DB2UTIL. These sections show how you use each type
and how each type uses the SAS data set to make a change in the DB2 table.
Inserting Data
You can insert observations from a SAS data set into a DB2 table as rows in the table. To use this insert function, name the SAS data set and the DB2 table in the PROC DB2UTIL statement. You can then use the MAPTO statement to map values from SAS variables to columns in the DB2 table. If you do not want to insert the values for all variables in the SAS data set into the DB2 table, map only the variables that you want to insert. However, you must map all DB2 columns to a SAS column.

Updating Data
You can change the values in DB2 table columns by replacing them with values from a SAS data set. You can change a column value to another value for every row in the table, or you can change column values only when certain criteria are met. For example, you can change the value of the NUM DB2 column to 10 for every row in the table. You can also change the value of the NUM DB2 column to the value in the NUMBER SAS variable if the DB2 column name value and the SAS data set variable name match.

You specify the name of the SAS data set and the DB2 table to be updated when you execute PROC DB2UTIL. You can specify that only certain variables be updated by naming only those variables in your mapping specifications.

You can use the WHERE clause to specify that only the rows on the DB2 table that meet certain criteria are updated. For example, you can use the WHERE clause to specify that only the rows with a certain range of values are updated. Or you can specify that rows to be updated when a certain column value in the row matches a certain SAS variable value in the SAS data set. In this case, you could have a SAS data set with several observations in it. For each observation in the data set, the DB2UTIL procedure updates the values for all rows in the DB2 table that have a matching value. The procedure then goes on to the next observation in the SAS data set and continues to update values in DB2 columns in rows that meet the comparison criteria.

Deleting Data
You can remove rows from a DB2 table when a certain condition is met. You can delete rows from the table when a DB2 column value in the table matches a SAS variable value in the SAS data set. Name the DB2 table from which you want to delete rows and the SAS data set that contains the target deletion values in the PROC DB2UTIL statement. You can then use the WHERE statement to specify the DB2 column name and the SAS variable whose values must match before the deletion is performed. To delete values that are based on criteria other than values in SAS data variables, you can use an SQL DELETE statement. For example, you can delete every row with a department number of 600.

PROC DB2UTIL Example
This example uses the UPDATE function in PROC DB2UTIL to update a list of telephone extensions from a SAS data set. The master list of extensions is in the DB2 table Testid.Employees and is updated from the SAS data set Trans. First, create the SAS data set.

```sas
options sastrace=',,d';

data trans;
  empno=321783;
  ext='3999';
  output;
  empno=320001;
  ext='4321';
```

The DB2UTIL Procedure 553
Next, specify the data set in PROC DB2UTIL.

```
proc db2util data=trans table=testid.employees function=u;
  mapto ext=phone;
  where empid=%empno;
  update;
run;
```

The row that includes EMPID=320001 is not found in the Testid.Employees table and is therefore not updated. You can ignore the warning in the SAS log.

---

**Maximizing DB2 under z/OS Performance**

**Assessing When to Tune Performance**

Among the factors that affect DB2 performance are the size of the table that is being accessed and the form of the SQL SELECT statement. If the table that is being accessed is larger than 10,000 rows (or 1,000 pages), you should evaluate all SAS programs that access the table directly. When you evaluate the programs, consider these questions.

- Does the program need all columns that the SELECT statement retrieves?
- Do the WHERE clause criteria retrieve only those rows that are needed for subsequent analysis?
- Is the data going to be used by more than one procedure in one SAS session? If so, consider extracting the data into a SAS data file for SAS procedures to use instead of allowing the data to be accessed directly by each procedure.
- Do the rows need to be in a particular order? If so, can an indexed column be used to order them? If there is no index column, is DB2 doing the sort?
- Do the WHERE clause criteria allow DB2 to use the available indexes efficiently?
- What type of locks does DB2 need to acquire?
- Are the joins being passed to DB2?
- Can your DB2 system use parallel processing to access the data more quickly?

**When the Resource Limit Facility Limits Execution Time**

The DB2 Resource Limit Facility limits execution time of dynamic SQL statements. If the time limit is exceeded, the dynamic statement is terminated and the SQL code -905 is returned. This list describes several situations in which the RLF could stop a user from consuming large quantities of CPU time.

- an extensive join of DB2 tables with the SAS SQL procedure
- an extensive search by the FSEDIT, FSVIEW, or FSBROWSE procedures or an SCL application
- any extensive extraction of data from DB2
• an extensive select
• an extensive load into a DB2 table (In this case, you can break up the load by lowering the commit frequency, or you can use the bulk-load facility on page 561 through SAS/ACCESS Interface to DB2 under z/OS.)

Methods for Improving Performance

You can do several things in your SAS application to improve DB2 engine performance.

• Specify the SAS system option $SASTRACE='\',\',d'$. This option prints to the SAS log the dynamic SQL that the DB2 engine generated and all other SQL that the DB2 engine executed. You can then verify that all WHERE clauses, PROC SQL joins, and ORDER BY clauses are being passed to DB2. This option is for debugging purposes and should not be set once the SAS application is used in production. Specify $SASTRACE=OFF$ to disable this option.

• Verify that all SAS procedures and DATA steps that read DB2 data share connections where possible. You can do this by using one libref to reference all SAS applications that read DB2 data and by accepting the default value of SHAREDREAD for the CONNECTION= option.

• If your DB2 subsystem supports parallel processing, you can assign a value to the CURRENT DEGREE special register. Setting this register might enable your SQL query to use parallel operations. You can set the special register by using the LIBNAME options DBCONINIT= or DBLIBINIT= with the SET statement as shown in this example:

libname mydb2 db2 dbconinit="SET CURRENT DEGREE='ANY'";

• Use the view descriptor WHERE clause or the DBCONDITION= option to pass WHERE clauses to DB2. You can also use these methods to pass sort operations to DB2 with the ORDER BY clause instead of performing a sort within SAS.

• If you are using a SAS application or an SCL application that reads the DB2 data twice, let the DB2 engine spool the DB2 data. This happens by default because the default value for the SPOOL= option is YES.

The spool file is read both when the application rereads the DB2 data and when the application scrolls forward or backward through the data. If you do not use spooling but need to scroll backward through the DB2 table, the DB2 engine must start reading from the beginning of the data to the row that you want to access.

• Use the SQL procedure to pass joins to DB2 instead of using the MATCH MERGE capability (that is, merging with a BY statement) of the DATA step.

• Use the DBKEY= option when you are doing SAS processing that involves the KEY= option. When you use the DBKEY= option, the DB2 engine generates a WHERE clause that uses parameter markers. During the execution of the application, the values for the key are substituted into the parameter markers in the WHERE clause.

If you do not use the DBKEY= option, the entire table is retrieved into SAS, and the join is performed in SAS.

• Consider using stored procedures when they can improve performance in client/server applications by reducing network traffic. You can execute a stored procedure by using the DBCONINIT= or DBLIBINIT= LIBNAME options.

• Use the READBUFF= LIBNAME option to retrieve records in blocks instead of one at a time.
Optimizing Your Connections

The DB2 engine supports more than one connection to DB2 per SAS session. This enables you to separate tasks that fetch rows from a cursor from tasks that must issue commits. This separation eliminates having to resynchronize the cursor, prepare the statement, and fetch rows until you are positioned back on the row that you were on. This separation enables commit tasks to avoid locking contention and act sooner because they are not required to wait for cursors to close before resynchronizing. In general, tables that are opened for input fetch from cursors do not issue commits, although update openings might, and output openings do, issue commits.

You can control how the DB2 engine uses connections by using the CONNECTION= option in the LIBNAME statement. At one extreme is CONNECTION=UNIQUE, which causes each table access, whether it is for input, update, or output, to create and use its own connection. Conversely, CONNECTION=SHARED means that only one connection is made, and that input, update, and output accesses all share that connection.

The default value for the CONNECTION= option is CONNECTION=SHAREDREAD, which means that tables opened for input share one connection. Update and output openings obtain their own connections. CONNECTION=SHAREDREAD allows for the best separation between tasks that fetch from cursors and tasks that must issue commits, eliminating the resynchronizing of cursors.

The values GLOBAL and GLOBALREAD perform similarly to SHARED and SHAREDREAD. The difference is that you can share the given connection across any of the librefs that you specify as GLOBAL or GLOBALREAD.

Although the default value of CONNECTION=SHAREDREAD is usually optimal, at times another value might be better. If you must use multiple librefs, you might want to set them each as GLOBALREAD. In this case, you have one connection for all of your input openings, regardless of which libref you use, as opposed to one connection per libref for input openings. In a single-user environment (as opposed to a server session), you might know that you do not have multiple openings occurring at the same time. In this case, you might want to use SHARED or GLOBAL for multiple librefs. By using such a setting, you eliminate the overhead of creating separate connections for input, update, and output transactions. If you have only one opening at a time, you eliminate the problem of resynchronizing input cursors if a commit occurs.

Another reason for using SHARED or GLOBAL is the case of opening a table for output while opening another table within the same database for input. This can result in a -911 deadlock situation unless both opens occur in the same connection.

As explained in “DB2 under z/OS Information for the Database Administrator” on page 578, the first connection to DB2 is made from the main SAS task. Subsequent connections are made from corresponding subtasks, which the DB2 engine attaches; DB2 allows only one connection per task. Due to the system overhead of intertask communication, the connection established from the main SAS task is a faster connection in terms of CPU time. Because this is true, you can expect better performance (less CPU time) if you use the first connection for these operations when you read or write large numbers of rows. If you read-only rows, SHAREDREAD or GLOBALREAD can share the first connection. However, if you are both reading and writing rows (input and output opens), you can use CONNECTION=UNIQUE to make each opening use the first connection. UNIQUE causes each opening to have its own connection. Suppose you have only one opening at a time and some are input while others are output for large amounts of data. The performance benefit of using the main SAS task connection far outweighs the overhead of establishing a new connection for each opening.
The utility connection is another type of connection that the DB2 engine uses, which the use does not control. This connection is a separate connection that can access the system catalog and issue commits to release locks. Utility procedures such as DATASETS and CONTENTS can cause this connection to be created, although other actions necessitate it as well. There is one connection of this type per libref, but it is not created until it is needed. If you have critical steps that must use the main SAS task connection for performance reasons, refrain from using the DEFER=YES option in the LIBNAME statement. It is possible that the utility connection can be established from that task, causing the connection that you use for your opening to be from a slower subtask.

In summary, no single value works best for the CONNECTION= option in all possible situations. You might need to try different values and arrange your SAS programs in different ways to obtain the best performance possible.

### Passing SAS Functions to DB2 under z/OS

SAS/ACCESS Interface to DB2 under z/OS passes the following SAS functions to DB2 for processing if the DBMS driver or client that you are using supports this function. Where the DB2 function name differs from the SAS function name, the DB2 name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

- ABS
- MOD (see note)
- ARCCOS (ACOS)
- MONTH
- ARSIN (ASIN)
- QTR (QUARTER)
- ATAN
- RIGHT (RTRIM)
- ATAN2
- SECOND
- AVG
- SIGN
- CEIL
- SIN
- COS
- SINH
- COSH
- SQRT
- COUNT
- STD (STDDEV)
- DAY
- STRIP
- EXP
- SUBSTR
- FLOOR
- SUM
- HOUR
- TAN
- INDEX (LOCATE)
- TANH
- LEFT (LTRIM)
- TRANWRD (REPLACE)
- LOWCASE (LCASE)
- TRIMN (RTRIM)
- LOG
- TRUNC
- LOG10
- UPCASE (UCASE)
- MAX
- VAR (VARIANCE)
- MIN
- WEEKDAY (DAYOFWEEK)
- MINUTE
- YEAR

**Note:** SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.
SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to DB2. Due to incompatibility in date and time functions between DB2 and SAS, DB2 might not process them correctly. Check your results to determine whether these functions are working as expected.

<table>
<thead>
<tr>
<th>SAS Functions</th>
<th>Database Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATEPART (DATE)</td>
<td>TIMEPART (TIME)</td>
</tr>
<tr>
<td>LENGTH</td>
<td>TODAY (CURRENT DATE)</td>
</tr>
<tr>
<td>REPEAT</td>
<td>TRANSLATE</td>
</tr>
</tbody>
</table>

These functions are not passed to the DBMS in DB2 V5. None of them existed in DB2 prior to DB2 V6. Also, there is no way to determine what location you are connected to and which functions are supported when you connect using DRDA.

The following functions are passed to the DBMS in DB2 V5, V6, and later. They are not passed to the DBMS when you connect using DRDA.

- YEAR
- MONTH
- DAY

---

**Passing Joins to DB2 under z/OS**

With these exceptions, multiple libref joins are passed to DB2 z/OS.

- If you specify the SERVER= option for one libref, you must also specify it for the others, and its value must be the same for all librefs.
- If you specify the DIRECT_SQL= option for one or multiple librefs, you must not set it to NO, NONE, or NOGENSQL.

For completeness, the portable code checks these options, regardless of the engine:

- DBCONINIT=
- DBCONTERM=
- DBLIBINIT=
- DBLIBTERM=
- DIRECT_EXE=
- DIRECT_SQL=
- PRESERVE_COL_NAMES=
- PRESERVE_TAB_NAMES=PRESERVE_TAB_NAMES=

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.
You can use these SAS system options when you start a SAS session that accesses DB2 under z/OS.

**DB2DECPT=decimal-value**  
specifies the setting of the DB2 DECPOINT= option. The decpoint-value argument can be a period (.) or a comma (,). The default is a period (.).  

DB2DECPT= is valid as part of the configuration file when you start SAS.

**DB2IN='database-name.table-space-name' | 'DATABASE database-name'**  
lets you specify the database and table space in which you want to create a new table. The DB2IN= option is relevant only when you are creating a new table. If you omit this option, the default is to create the table in the default database and table space.  

database.table-space specifies the names of the database and table space.  

'DATABASE database-name' specifies only the database name. Enclose the entire specification in single quotation marks.

You can override the DB2IN= system option with the IN= LIBNAME or data set option.

**DB2PLAN=plan-name**  
specifies the name of the plan that is used when connecting (or binding) SAS to DB2. SAS provides and supports this plan, which can be adapted for each user's site. The value for DB2PLAN= can be changed at any time during a SAS session, so that different plans can be used for different SAS steps. However, if you use more than one plan during a single SAS session, you must understand how and when SAS/ACCESS Interface to DB2 under z/OS makes the connections. If one plan is in effect and you specify a new plan, the new plan does not affect the existing DB2 connections.

**DB2RRS | NODB2RRS**  
specifies the attachment facility to be used for a SAS session when connecting to DB2. This option is an invocation-only option.

Specify NODB2RRS, the default, to use the Call Attachment Facility (CAF). Specify DB2RRS to use the Recoverable Resource Manager Services Attachment Facility (RRSAF). For details about using RRSAF, see “How the Interface to DB2 Works” on page 578.

**DB2RRSMP | NODB2RRSMP**  
specifies that the multiphase SRRCMIT commit and SRRBACK rollback calls are used instead of the COMMIT and ROLLBACK SQL statements. This option is ignored unless DB2RRS is specified. This option is available only at invocation.

Specify NODB2RRSMP, the default, when DB2 is the only Resource Manager for your application. Specify DB2RRSMP when your application has other resource managers, which requires the use of the multiphase calls. Using the multiphase calls when DB2 is your only resource manager can have performance implications. Using COMMIT and ROLLBACK when you have more than one resource manager can result in an error, depending on the release of DB2.
DB2SSID=subsystem-name
specifies the DB2 subsystem name. The subsystem-name argument is one to four
characters that consist of letters, numbers, or national characters (#, $, or @); the first
character must be a letter. The default value is DB2. For more information, see
“Settings” on page 560.

DB2SSID= is valid in the OPTIONS statement, as part of the configuration file, and
when you start SAS.

You can override the DB2SSID= system option with the SSID= connection option.

DB2UPD=Y | N
specifies whether the user has privileges through SAS/ACCESS Interface to DB2
under z/OS to update DB2 tables. This option applies only to the user's Update
privileges through the interface and not necessarily to the user's privileges while
using DB2 directly. Altering the setting of DB2UPD= has no effect on your DBMS
privileges, which have been set with the GRANT statement. The default is Y (Yes).

DB2UPD= is valid in the OPTIONS statement, as part of the configuration file, and
when you start SAS. This option does not affect PROC DBLOAD or the SAS 5
compatibility procedures.

Settings

To connect to DB2, you must specify a valid DB2 subsystem name in one of these ways.

- the DB2SSID= system option (SAS/ACCESS Interface to DB2 under z/OS uses this
  value if no DB2 subsystem is specified.)

- the SSID= option in the PROC ACCESS statement

- the SSID= statement of PROC DBLOAD

- the SSID= option in the PROC SQL CONNECT statement, which is part of the SQL
  pass-through facility

- the SSID= connection option in the LIBNAME statement

If a site does not specify a valid DB2 subsystem when it accesses DB2, this message is
generated:

ERROR: Cannot connect to DB2 subsystem XXXX,
r=12, reason code = 00F30006. See the
Call Attachment Facility documentation
for an explanation.

XXX is the name of the subsystem to which SAS tried to connect. To find the correct
value for your DB2 subsystem ID, contact your database administrator.

Macros

Use the automatic SYSDBRC macro variable to capture DB2 return codes when using
the DB2 engine. The macro variable is set to the last DB2 return code that was
encountered only when execution takes place through SAS/ACCESS Interface to DB2
under z/OS. If you reference SYSDBRC before engine processing takes place, you
receive this message:

WARNING: Apparent symbolic reference SYSDBRC not resolved.
Use SYSDBRC for conditional post-processing. Below is an example of how to abend a job. The table DB2TEST is dropped from DB2 after the view descriptor is created, resulting in a -204 code.

```
data test;
x=1;
y=2;
proc dbload dbms=db2 data=test;
table=db2test;
in 'database test';
load;
run;
```

```
proc access dbms=db2;
create work.temp.access;
table=user1.db2test;
create work.temp.view;
select all;
run;
proc sql;
execute(drop table db2test)by db2;
quit;
```

```
proc print data=temp;
run;
```

```
data _null_; if "&sysdbrc" not in ('0','100') then do;
   put 'The DB2 Return Code is: ' "&sysdbrc";
   abort abend;
end;
run;
```

Because the abend prevents the log from being captured, you can capture the SAS log by using the SAS system option, ALTLOG.

---

**Bulk Loading for DB2 under z/OS**

**Overview**

By default, the DB2 under z/OS interface loads data into tables by preparing an SQL INSERT statement, executing the INSERT statement for each row, and issuing a COMMIT statement. You must specify BULKLOAD=YES to start the DB2 LOAD utility. You can then bulk load rows of data as a single unit, which can significantly enhance performance. For smaller tables, the extra overhead of the bulk-loading process might slow performance. For larger tables, the speed of the bulk-loading process outweighs the overhead costs.

When you use bulk loading, see the SYSPRINT output for information about the load. If you run the LOAD utility and it fails, ignore the messages in the SAS log because they might be inaccurate. However, if errors existed before you ran the LOAD utility, error messages in the SAS log might be valid.
SAS/ACCESS Interface to DB2 under z/OS provides bulk loading through DSNUTILS, an IBM stored procedure that starts the DB2 LOAD utility. DSNUTILS is included in DB2 Version 6 and later, and it is available for DB2 Version 5 in a maintenance release. Because the LOAD utility is complex, familiarize yourself with it before you use it through SAS/ACCESS. Also check with your database administrator to determine whether this utility is available.

**Data Set Options for Bulk Loading**

The DB2 under z/OS engine supports the following bulk-load data set options. All begin with BL_ for bulk loading. To use the bulk-load facility, you must specify `BULKLOAD=YES` or all bulk-load options are ignored. (The DB2 under z/OS interface alias for BULKLOAD= is DB2LDUTIL=.)

- **BL_DB2CURSOR=**
- **BL_DB2DATACLAS=**
- **BL_DB2DEVT_PERM=**
- **BL_DB2DEVT_TEMP=**
- **BL_DB2DISC=**
- **BL_DB2ERR=**
- **BL_DB2IN=**
- **BL_DB2LDCT1=**
- **BL_DB2LDCT2=**
- **BL_DB2LDCT3=**
- **BL_DB2LDEXT=**
- **BL_DB2MAP=**
- **BL_DB2MGMTCLAS=**
- **BL_DB2PRINT=**
- **BL_DB2PRNLOG=**
- **BL_DB2REC=**
- **BL_DB2RECS=**
- **BL_DB2RSTRT=**
- **BL_DB2SPC_PERM=**
- **BL_DB2SPC_TEMP=**
- **BL_DB2STORCLAS=**
- **BL_DB2TBLXST=**
- **BL_DB2UNITCOUNT=**
- **BL_DB2UTID=**
- **BULKLOAD=**

**File Allocation and Naming for Bulk Loading**

When you use bulk loading, these files (data sets) are allocated.
• The DB2 DSNUTILS procedure allocates these as new and catalogs the SysDisc, SysMap, and SysErr files unless BL_DB2LDEXT=USERUN. If BL_DB2LDEXT=USERUN, data sets are allocated as old and are kept.

• The DB2 interface engine allocates as new and catalogs the files SysIn and SysRec when the execution method specifies to generate them.

• The DB2 interface engine allocates as new and catalogs the file SysPrint when the execution method specifies to run the utility.

All allocations of these data sets are reversed by the end of the step. If errors occur before SysRec is generated, any of these data sets that were allocated as new and cataloged are deleted as part of cleanup because they would be empty.

The interface engine uses these options when it allocates nonexisting SYS data set names.

• DSNUTILS uses BL_DB2DEVT_PERM= and BL_DB2SPC_PERM= for SysDisc, SysMap, and SysErr.

• The DB2 interface engine uses BL_DB2DEVT_PERM= for SysIn, SysRec, and SysPrint.

• SysRec uses BL_DB2RECSPC=. BL_DB2RECSPC= is necessary because the engine cannot determine how much space the SysRec requires—it depends on the volume of data being loaded into the table.

• DSNUTILS uses BL_DB2DEVT_TEMP= and BL_DB2SPC_TEMP= to allocate the other data set names that the LOAD utility requires.

This table shows how SysIn and SysRec are allocated based on the values of BL_DB2LDEXT= and BL_DB2IN=, and BL_DB2REC=.

![Table 17.3 SysIn and SysRec Allocation](image)

When SAS/ACCESS Interface to DB2 under z/OS uses existing files, you must specify the filenames. When the interface generates the files, it creates them with names that you provide or with unique names that it generates. Engine-generated filenames use system-generated data set names with the format

SYSyyddd.Thhmss.RA000.jobname.name.Hgg, where

SYSyyddd

is replaced by the user ID. The user ID that is used to prequalify these generated data set names is determined the same way as within the rest of SAS, except when
running in a server environment. In a server environment, the authenticated ID of the
client is used.

`name`

is replaced by the given SYS ddname of the data set.

For example, if you do not specify any data set names and run GENRUN under TSO,
you obtain a set of files allocated with names such as

```
USERID.T125547.RA000.USERID.DB2DISC.H01
USERID.T125547.RA000.USERID.DB2ERR.H01
USERID.T125547.RA000.USERID.DB2IN.H01
USERID.T125547.RA000.USERID.DB2MAP.H01
USERID.T125547.RA000.USERID.DB2PRINT.H01
USERID.T125547.RA000.USERID.DB2REC.H01
```

This naming convention produces unique names, even within a sysplex (within one
second per user ID per system). It therefore makes it easy to associate all information
for each utility execution and separate it from other executions.

Bulk-load files are removed at the end of the load process to save space. They are not
removed if the utility fails to allow for the load process to be restarted.

**Examples**

Use these LIBNAME statements for all examples.

```
libname db2lib db2;
libname shlib db2 connection=shared;
```

Create a table.

```
data db2lib.table1 (bulkload=yes);
  x=1;
  name='Tom';
run;
```

Append Table1 to itself.

```
data shlib.table1
  (bulkload=yes bl_db2tblxst=yes bl_db2ldct1='RESUME YES');
  set shlib.table1;
run;
```

Replace Table1 with itself.

```
data shlib.table1
  (bulkload=yes bl_db2tblxst=yes bl_db2ldct1='REPLACE');
  set shlib.table1;
run;
```

Load DB2 tables directly from other objects.

```
data db2lib.emp (bulkload=yes
  bl_db2ldct1='REPLACE LOG NO NOCOPYPEND'
  bl_db2cursor='select * from dsn8710.emp');
  set db2lib.emp (obs=0);
run;
```

You can also use this option in a PROC SQL statement to load DB2 tables directly from
other objects, as shown below.

```
options sastrace=',,d';
```
libname db2lib db2 authid=dsn8710;
libname mylib db2;

proc datasets library=mylib;
  delete emp;run;

proc sql;
  connect to db2;
  create table mylib.emp
  (BULKLOAD=YES
   BL_DB2LDCT1='REPLACE LOG NO NOCOPYPEND'
   BL_DB2CURSOR='SELECT FIRSTNAME, LASTNAME, WORKDEPT,
                 HIREDATE, JOB, SALARY, BONUS, COMM
                 FROM DSN8710.EMP')
  as select firstname, lastname, workdept,
           hiredate, job, salary, bonus, comm
  from db2lib.emp (obs=0);
quit;

Here is another similar example.

options sastrace=',,,d';
libname db2lib db2 authid=dsn8710;
libname mylib db2;

proc datasets library=mylib;
  delete emp;run;

proc sql;
  connect to db2;
  create table mylib.emp
  (BULKLOAD=YES
   BL_DB2LDCT1='REPLACE LOG NO NOCOPYPEND'
   BL_DB2CURSOR='SELECT FIRSTNAME, LASTNAME, WORKDEPT,
                 HIREDATE, JOB, SALARY, BONUS, COMM
                 FROM DSN8710.EMP'
   BL_DB2LDCT3='RUNSTATS TABLESPACE DSNDB04.TEMPTTABL
                 TABLE(ALL) INDEX(ALL) REPORT YES')
  as select firstname, lastname, workdept,
           hiredate, job, salary, bonus, comm
  from db2lib.emp (obs=0);
quit;

Generate control and data files, create the table, but do not run the utility to load it.

data shlib.table2 (bulkload=yes
  bl_db2ldext=genonly bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
  set shlib.table1;
run;

Use the control and data files that you generated in the preceding example load the table. The OBS=1 data set option on the input file prevents the DATA step from reading the whole file. Because the data is really in SysRec, you need only the input file to satisfy the engine.

data db2lib.table2 (bulkload=yes bl_db2tblxst=yes
  bl_db2ldext=userun bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
  set db2lib.table1 (obs=1);
run;
A more efficient approach than the previous example is to eliminate going to DB2 to read even one observation from the input table. This also means that the DATA step processes only one observation, without any input I/O. Note that the one variable V is not on the table. Any variables listed here (there is no need for more than one), are irrelevant because the table already exists; they are not used.

```sas
data db2lib.table2 (bulkload=yes bl_db2tblxst=yes
  b1_db2ldext=userid.sysin' bl_db2rec='userid.sysrec');
v=0;
run;
```

Generate control and data files, but do not create the table or run the utility. Setting BL_DB2TBLXST=YES when the table does not exist prevents you from creating the table; this only makes sense because you are not going to load any data into the table at this time.

```sas
data db2lib.table3 (bulkload=yes bl_db2tblxst=yes
  b1_db2ldext=genonly bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
set db2lib.table1;
run;
```

Use the control and data files that you generated in the preceding example to load the table. The OBS=1 data set option on the input file prevents the DATA step from reading the whole file. In this case, you must specify the input file because it contains the column definitions that are necessary to create the table.

```sas
data shlib.table3 (bulkload=yes bl_db2ldext=userun
  b1_db2in='userid.sysin' bl_db2rec='userid.sysrec');
set shlib.table1 (obs=1);
run;
```

If you know the column names, a more efficient approach than the previous example is to eliminate going to DB2 to obtain the column definitions. In this case, the variable names and data types must match, because they are used to create the table. However, the values specified for the variables are not included on the table, because all data to load comes from the existing SysRec.

```sas
data db2lib.table3 (bulkload=yes bl_db2ldext=userid
  b1_db2in='userid.sysin' bl_db2rec='userid.sysrec');
x=0;
  name='???';
run;
```

You can use other applications that do output processing.

```sas
data work.a;
x=1;
run;
```

```sas
proc sql;
  create db2lib.table4 (bulkload=yes) as select * from a;
quit;
```

---

### Locking in the DB2 under z/OS Interface

The following LIBNAME and data set options let you control how the DB2 under z/OS interface handles locking. For general information about an option, see "LIBNAME"
Statement Syntax for Relational Databases” on page 97. For additional information, see your DB2 documentation.

READ_LOCK_TYPE=TABLE

UPDATE_LOCK_TYPE=TABLE

READ_ISOLATION_LEVEL= CS | UR | RR | "RR KEEP UPDATE LOCKS" | RS | "RS KEEP UPDATE LOCKS"

Here are the valid values for this option. DB2 determines the default isolation level.

Table 17.4 Isolation Levels for DB2 under z/OS

<table>
<thead>
<tr>
<th>Value</th>
<th>Isolation Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Cursor stability</td>
</tr>
<tr>
<td>UR</td>
<td>Uncommitted read</td>
</tr>
<tr>
<td>RR</td>
<td>Repeatable read</td>
</tr>
<tr>
<td>RR KEEP UPDATE LOCKS*</td>
<td>Repeatable read keep update locks</td>
</tr>
<tr>
<td>RS</td>
<td>Read stability</td>
</tr>
<tr>
<td>RS KEEP UPDATE LOCKS*</td>
<td>Read stability keep update locks</td>
</tr>
</tbody>
</table>

* When specifying a value that consists of multiple words, enclose the entire string in quotation marks.

UPDATE_ISOLATION_LEVEL= CS | UR | RR | "RR KEEP UPDATE LOCKS" | RS | "RS KEEP UPDATE LOCKS"

The valid values for this option are described in the preceding table. The default isolation level is determined by DB2.

---

Naming Conventions for DB2 under z/OS

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Although DB2 is case sensitive, it converts table and column names to uppercase by default. To preserve the case of the table and column names that you send to DB2, enclose them in quotation marks.

DB2 objects include tables, views, columns, and indexes. They follow these naming conventions.

- These objects must have names of the following length in characters: column (1–30), index (1–128), table (1–128), view (1–128), alias (1–128), synonym (1–128), correlation (1–128). However, SAS limits table names to 32 bytes. This limitation prevents database table objects that are defined through a DATA step—for example, to have names that are longer than 32.
These objects must have names from 1–8 characters long: authorization ID, referential constraint, database, table space, storage group, package, or plan.

A location name can be 1–16 characters long.

- A name must begin with a letter. If the name is in quotation marks, it can start with and contain any character. Depending on how your string delimiter is set, quoted strings can contain quotation marks (for example, “O’Malley”).
- A name can contain the letters A–Z, numbers from 0–9, number or pound sign (#), dollar sign ($), or at symbol (@).
- Names are not case sensitive. For example, CUSTOMER and Customer are the same. However, if the name of the object is in quotation marks, it is case sensitive.
- A name cannot be a reserved word in DB2.
- A name cannot be the same as another DB2 object. For example, each column name within the same table must be unique.

---

**Data Types for DB2 under z/OS**

**Overview**

Every column in a table has a name and a data type. The data type tells DB2 how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about DB2 data types, NULL and default values, and data conversions.

For more information about DB2 data types, see your DB2 SQL reference documentation.

SAS/ACCESS does not support ROWID and some types of distinct DB2 data types.

**Supported Data Types for DB2 under z/OS**

Here are the data types that DB2 under z/OS supports:

- **Character data:**
  
  CHAR(n)  
  CLOB (character large object)  
  DBCLOB (double-byte character large object)  
  GRAPHIC(n)  
  LONG VARCHAR  
  LONG VARGRAPHIC  
  VARCHAR(n)  
  VARGRAPHIC(n)

- **Numeric data:**
  
  DECIMAL(p,s) | DEC(p,s)  
  INTEGER | INT, REAL | FLOAT(n)  
  FLOAT(n) | DOUBLE PRECISION | FLOAT | DOUBLE  
  SMALLINT  

Even though the DB2 numeric columns have these distinct data types, the DB2 engine accesses, inserts, and loads all numerics as FLOATs.

*Note:* When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that
contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

• Date, time, and timestamp data:

\[
\begin{align*}
\text{DATE} & \quad \text{TIMESTAMP} \\
\text{TIME} &
\end{align*}
\]

DB2 date and time data types are similar to SAS date and time values in that they are stored internally as numeric values and are displayed in a site-chosen format. The DB2 data types for dates, times, and timestamps are listed here. Note that columns of these data types might contain data values that are out of range for SAS, which handles dates from 1582 A.D. through 20,000 A.D.

• Binary data: BLOB (binary large object)

\textit{Note:} Support for large object data was added in the second maintenance release for SAS 9.4.

• ROWID data type

For more information about DB2 data types, see your DB2 documentation.

**DB2 Null and Default Values**

DB2 has a special value that is called NULL. A DB2 NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DB2 NULL value, it interprets it as a SAS missing value.

DB2 columns can be defined so that they do not allow NULL data. For example, NOT NULL would indicate that DB2 does not allow a row to be added to the TestID.Customers table unless there is a value for CUSTOMER. When creating a DB2 table with SAS/ACCESS, you can use the \texttt{DBNULL=} data set option to indicate whether NULL is a valid value for specified columns.

You can also define DB2 columns as NOT NULL WITH DEFAULT. The following table lists default values that DB2 assigns to columns that you define as NOT NULL WITH DEFAULT. An example of such a column is STATE in Testid.Customers. If a column is omitted from a view descriptor, default values are assigned to the column. However, if a column is specified in a view descriptor and it has no values, no default values are assigned.

\textit{Table 17.5}  \textit{Default Values That DB2 Assigns for Columns Defined as NOT NULL WITH DEFAULT}

<table>
<thead>
<tr>
<th>DB2 Column Type</th>
<th>DB2 Default*</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CHAR(n)}</td>
<td>blanks, unless the NULLCHARVAL= option is specified</td>
</tr>
<tr>
<td>\texttt{VARCHAR}</td>
<td>empty string</td>
</tr>
<tr>
<td>\texttt{LONG VARCHAR}</td>
<td></td>
</tr>
<tr>
<td>\texttt{VARGRAPHIC}</td>
<td></td>
</tr>
<tr>
<td>\texttt{LONG VARGRAPHIC}</td>
<td></td>
</tr>
</tbody>
</table>
DB2 Column Type | DB2 Default
---|---
BLOB | CLOB | DBCLOB | empty string
SMALLINT | INT | FLOAT | DECIMAL | REAL | 0
DATE | current date, derived from the system clock
TIME | current time, derived from the system clock
TIMESTAMP | current timestamp, derived from the system clock

* The default values that are listed in this table pertain to values that DB2 assigns.

Knowing whether a DB2 column allows NULL values or whether DB2 supplies a default value can assist you in writing selection criteria and in entering values to update a table. Unless a column is defined as NOT NULL or NOT NULL WITH DEFAULT, the column allows NULL values.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how the DBMS handles SAS missing character values, use the NULLCHAR= and NULLCHARVAL= data set options.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to DB2 assigns to SAS variables when using the **LIBNAME statement** to read from a DB2 table. These default formats are based on DB2 column attributes.

**Table 17.6 LIBNAME Statement: Default SAS Formats for DB2 Data Types**

<table>
<thead>
<tr>
<th>DB2 Column Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)*</td>
<td>$w:</td>
</tr>
<tr>
<td>VARCHAR(n)*</td>
<td>$HEX32767.</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>$32767.</td>
</tr>
<tr>
<td>BLOB</td>
<td>$HEX32767.</td>
</tr>
<tr>
<td>CLOB</td>
<td>$32767.</td>
</tr>
<tr>
<td>DBCLOB</td>
<td>$w:</td>
</tr>
<tr>
<td>GRAPHIC(n)*</td>
<td>$w:( w&lt;=127)</td>
</tr>
<tr>
<td>VARGRAPHIC(n)*</td>
<td>$127. (w&gt;127)</td>
</tr>
<tr>
<td>LONG VARGRAPHIC</td>
<td>$11.</td>
</tr>
<tr>
<td>DB2 Column Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>6.</td>
</tr>
<tr>
<td>DECIMAL(m,n)**</td>
<td>w+2.d</td>
</tr>
<tr>
<td>FLOAT</td>
<td>none</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>none</td>
</tr>
<tr>
<td>REAL</td>
<td>none</td>
</tr>
<tr>
<td>NUMERIC(m,n)**</td>
<td>w+2.d</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME8.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME30.6</td>
</tr>
<tr>
<td>ROWID</td>
<td>none</td>
</tr>
</tbody>
</table>

* n in DB2 character and graphic data types is equivalent to w in SAS formats.

** m and n in DB2 numeric data types are equivalent to w and d in SAS formats.

This table shows the default DB2 data types that SAS/ACCESS assigns to SAS variable formats during output operations.

**Table 17.7 LIBNAME Statement: Default DB2 Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>DB2 Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.</td>
<td>CHARACTER(n) for 1–255*</td>
</tr>
<tr>
<td>$CHARw.</td>
<td>VARCHAR(n) for &gt;255*</td>
</tr>
<tr>
<td>$VARYINGw.</td>
<td></td>
</tr>
<tr>
<td>$HEXw.</td>
<td></td>
</tr>
<tr>
<td>any date format</td>
<td>DATE</td>
</tr>
<tr>
<td>any time format</td>
<td>TIME</td>
</tr>
<tr>
<td>any datetime format</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>all other numeric formats</td>
<td>FLOAT</td>
</tr>
</tbody>
</table>

* n in DB2 character and graphic data types is equivalent to w in SAS formats.

**ACCESS Procedure Data Conversions**

This table shows the default SAS variable formats that SAS/ACCESS assigns to DB2 data types when you use the ACCESS procedure.
### Table 17.8   ACCESS Procedure: Default SAS Formats for DB2 Data Types

<table>
<thead>
<tr>
<th>DB2 Column Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR((n))</td>
<td>$w.\ (w\leq199)^\dagger$</td>
</tr>
<tr>
<td>VARCHAR((n))</td>
<td>$w.$</td>
</tr>
<tr>
<td></td>
<td>$200.\ (w&gt;200)^\dagger$</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>$w.$</td>
</tr>
<tr>
<td>BLOB</td>
<td>none</td>
</tr>
<tr>
<td>CLOB</td>
<td>none</td>
</tr>
<tr>
<td>DBCLOB</td>
<td>none</td>
</tr>
<tr>
<td>GRAPHIC((n))</td>
<td>$w.\ (w\leq127)^\dagger$</td>
</tr>
<tr>
<td>VARGRAPHIC((n))</td>
<td>$127.\ (w&gt;127)^\dagger$</td>
</tr>
<tr>
<td>LONG VARGRAPHIC</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>11.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>6.</td>
</tr>
<tr>
<td>DECIMAL((m,n))</td>
<td>( w+2.d )</td>
</tr>
<tr>
<td></td>
<td>For example, DEC(6,4) becomes SAS format 8.4</td>
</tr>
<tr>
<td>REAL</td>
<td>E12.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>E12.</td>
</tr>
<tr>
<td>FLOAT((n))</td>
<td>E12.</td>
</tr>
<tr>
<td>FLOAT</td>
<td>E12.</td>
</tr>
<tr>
<td>NUMERIC((m,n))</td>
<td>( w+2.d^\star )</td>
</tr>
<tr>
<td></td>
<td>For example, NUMERIC(6,2) becomes SAS format 8.2</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE7.</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME8.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME30.6</td>
</tr>
</tbody>
</table>

^ \dagger \ n in DB2 character and graphic data types is equivalent to \( w \) in SAS formats.

\star \ m and \ n in DB2 numeric data types are equivalent to \( w \) and \( d \) in SAS formats.

You can use the YEARCUTOFF= option to make your DATE7. dates comply with Year 2000 standards. For more information about this SAS system option, see *SAS System Options: Reference*. 
**DBLOAD Procedure Data Conversions**

This table shows the default DB2 data types that SAS/ACCESS assigns to SAS variable formats when you use the **DBLOAD procedure**.

**Table 17.9  DBLOAD Procedure: Default DB2 Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>DB2 Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.</td>
<td>CHARACTER(n)</td>
</tr>
<tr>
<td>$CHARw.</td>
<td></td>
</tr>
<tr>
<td>$VARYINGw.</td>
<td></td>
</tr>
<tr>
<td>$HEXw.</td>
<td></td>
</tr>
<tr>
<td>any date format</td>
<td>DATE</td>
</tr>
<tr>
<td>any time format</td>
<td>TIME</td>
</tr>
<tr>
<td>any datetime format</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>w.d</td>
<td>DECIMAL(m,n)†</td>
</tr>
<tr>
<td>IB, IBR, PIB, PIBR</td>
<td>INTEGER</td>
</tr>
<tr>
<td>all other numeric formats</td>
<td>FLOAT</td>
</tr>
</tbody>
</table>

* m and n in DB2 numeric data types are equivalent to w and d in SAS formats.

---

**Temporal Data for DB2 under z/OS**

**Overview of Temporal Data for DB2 under z/OS**

Temporal data stores information about values that change over time. Temporal data is structured to reduce the effort that is required to maintain and manage temporal columns in database tables. Using temporal data enables you to take advantage of simplified query syntax and semantics when working with records that contain data that pertains to specific time periods.

In DB2, there are three types of temporal data: system time, business time, and bitemporal data. **System time** tracks when changes are made to the state of data in a table. The state of a record in a table is determined by the SYS_START and SYS_END values. In a table that manages system time, the transaction time, or TRANS_START, is also maintained. The TRANS_START time records when a transaction to change a record, such as updating the SYS_END date, took place. SAS generates these three columns automatically when you create a table that stores system time data.

**Business time** tracks the effective dates for business data, such as promotional start and end dates. The effective start and end dates for business time data are stored in the BUS_START and BUS_END values. SAS generates these columns automatically when you create a table that stores business time data.
Bitemporal data tracks system time data and business time data. A table that contains bitemporal data contains all of the columns that are automatically included for system time data and business time data: SYS_START, SYS_END, TRANS_START, BUS_START, and BUS_END.

System Time and Bitemporal Data: History Tables

Tables that contain business time data include historical, current, and future data records. For tables that contain system time data or bitemporal data, SAS automatically generates history tables. The name of a history table is constructed as <base-table>_HISTORY. History tables contain records that are determined to be historical. That is, when the SYS_END date precedes the current date or datetime, then the record is a historical record. SAS/ACCESS for DB2 under z/OS automatically issues an ALTER TABLE ... ADD VERSIONING statement that links the base table with its history table.

The temporal table and its associated history table must be stored in separate table spaces. In addition, each table should be the only table in its table space. If a table space is not included as part of the creation of a temporal table from a SAS DATA step, then DB2 automatically assigns a unique table space to the temporal table and to the associated history table. For this reason, you cannot specify a table space in the values for the IN= option or the DB2IN= system option, if you are working with temporal tables that contain system time or bitemporal data. For more information, see IN= data set option, IN= LIBNAME option, or “SAS System Options, Settings, and Macros for DB2 under z/OS” on page 559.

Data Set Options for Temporal Data

The following data set options are used when you work with temporal data:

TEMPORAL=
   specifies the type of temporal data that is stored in a table. Possible values are BUSINESS, SYSTEM, or BITEMPORAL. For more information, see “TEMPORAL= Data Set Option” on page 412.

BUSINESS_DATATYPE=
   specifies the data type for the BUS_START and BUS_END values. Possible values are DATE or TIMESTAMP(6). The default value is TIMESTAMP(6). For more information, see “BUSINESS_DATATYPE= Data Set Option” on page 319.

BUSINESS_TIMEFRAME=
   specifies a time period to be used when querying or modifying a table that contains temporal data. You provide the beginning and ending values for the time period based on the BUSINESS_DATATYPE value. Specify the time period in the format:

   FROM <date-or-datetime> TO <date-or-datetime>

For example, use the following code to specify the time period from January 1, 2014 to December 31, 2014 (using date values):

   businesstimeframe="from '01JAN2014'd to '31DEC2014'd"

   For more information, see “BUSINESS_TIMEFRAME= Data Set Option” on page 319.

SYSTEM_TIMEFRAME=
   specifies a time period to be used when querying or modifying a table that contains temporal data. You provide the beginning and ending datetime values in the format:

   FROM <datetime> TO <datetime>
Provide datetime values that include year, month, day, hours, minutes, seconds, and fractions of a second. For example, use the following code to specify the time period from midnight, November 1, 2012 to midnight, December 1, 2030:

```
system_timeframe="from '2012-11-01-00.00.00.0' to '2030-12-01-00.00.00.0'"
```

For more information, see “SYSTEM_TIMEFRAME= Data Set Option” on page 410.

OVERLAPS=

specifies columns that should not contain active business time periods that overlap. By default, overlaps are allowed for active business time records. Separate column names with commas. For more information, see “OVERLAPS= Data Set Option” on page 388.

---

**Understanding DB2 under z/OS Client/Server Authorization**

**Libref Connections**

When you use the DB2 interface, you can enable each client to control its own connections using its own authority—instead of sharing connections with other clients—by using the DB2 Recoverable Resource Manager Services Attachment Facility (RRSAF). See “DB2 Attachment Facilities (CAF and RRSAF)” on page 580 for information about this facility.

When you use SAS/ACCESS Interface to DB2 under z/OS with RRSAF, the authorization mechanism works differently than it does in Base SAS:

- In Base SAS, the SAS server always validates the client's authority before allowing the client to access a resource.

- In SAS/ACCESS Interface to DB2 under z/OS (with RRSAF), DB2 checks the authorization identifier that is carried by the connection from the SAS server. In most situations, this is the client's authorization identifier. In one situation, however, this is the SAS server's authorization identifier. A client can access a resource by using the server's authorization identifier only if the client uses a libref that was predefined in the server session.

In this next example, a user assigns the libref SRVPRELIB in the SRV1 server session. In the client session, a user then issues a LIBNAME statement that makes a logical assignment using the libref MYPRELIB, and the user specifies the LIBNAME option SERVER=srv1. The client can then access resources by using the server's authority for the connection.

1. In the server session
   ```
   libname srvprelib db2 ssid=db25;
   proc server id=srv1;
   run;
   ```

2. In the client session
   ```
   libname myprelib server=srv1 slibref=srvprelib;
   proc print data=myprelib.db2table;
   run;
   ```
In this example, because the client specifies a regular libref, MYDBLIB, the client has its own authority for the connections.

1. In the server session
   
   libname myprelib db2 ssid=db25;
   proc server id=srv1;
   run;

2. In the client session
   
   libname mydblib server=srv1 roptions='ssid=db25' rengine=db2;
   proc print data=mydblib.db2table;
   run;

In this table, SAS/SHARE clients use LIBNAME statements to access SAS libraries and DB2 data through the server. In this description, a logical LIBNAME statement is a statement that associates a libref with another libref that was previously assigned.

### Table 17.10 Librefs and Their Authorization Implications

**Client Session**

<table>
<thead>
<tr>
<th>libname local v8 'SAS.library' disp=old;</th>
<th>These statements execute in the client session. these are local assignments. The authority ID is the ID of the client.</th>
</tr>
</thead>
<tbody>
<tr>
<td>libname dblocal db2 connection=unique;</td>
<td></td>
</tr>
<tr>
<td>libname remote 'SAS.library' server=serv1 rengine=v8 roptions='disp=old';</td>
<td>These statements execute in the server session on behalf of the client. Libref Remote is a Base SAS engine remote assignment. Libref DbRemote is a DB2 engine remote assignment. In both cases, the authority ID is the ID of the client.</td>
</tr>
<tr>
<td>libname dbremote server=serv1 rengine=db2 roptions='connection=unique';</td>
<td></td>
</tr>
</tbody>
</table>

**Server Session (id=serv1)**

<table>
<thead>
<tr>
<th>libname predef v8 'SAS.library' disp=old;</th>
<th>Because librefs PreDef and DbPreDef are defined in the server session, they can be referenced only by a client using a logical LIBNAME statement. There is no authority ID because clients cannot access these librefs directly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>libname dbpredef db2 connection=unique;</td>
<td></td>
</tr>
</tbody>
</table>

Logical Assignments - Client Session

<table>
<thead>
<tr>
<th>libname alias (local);</th>
<th>These statements create aliases ALIAS and DBALIAS for librefs Local and DbLocal, which were assigned in the client session above. The authority ID is the ID of the client.</th>
</tr>
</thead>
<tbody>
<tr>
<td>libname dbalias (dblocal);</td>
<td></td>
</tr>
</tbody>
</table>
libname logic server=serv1 slibref=predef;
libname dblogic server=serv1 slibref=dbpredef;

These statements refer to librefs PreDef and DbPreDef, which were assigned in the server session above.

Libref Logic is a Base SAS engine logical assignment of remote libref PreDef. The authority ID for libref Logic is the ID of the client.

Libref DbLogic is a DB2 engine logical assignment of remote libref DbPreDef. The authority ID for libref DbLogic is the ID of the server.

For the Base SAS engine Remote and Logic librefs, it is the client’s authority that is verified. (This is true for all Base SAS engine assignments.) DbRemote and DbLogic DB2 engine librefs refer to the same resources. However, it is the client’s authority that is verified for DbRemote, whereas it is the server’s authority that is verified for DbLogic. When using the DB2 interface, you can determine whether to use the client’s authority or the server’s authority to access DB2 data.

**Non-Libref Connections**

When you make connections using the SQL pass-through facility or view descriptors, the connections to the database are not based on a DB2 engine libref. A connection that is created in the server, by using these features from a client, always has the authority of the client, because there is no server-established connection to reference.

This example uses the SAS/SHARE Remote SQL pass-through facility. The client has its own authority for the connections.

1. In the server session:
   ```sas
   proc server id=srv1;
   run;
   ```

2. In the client session
   ```sas
   proc sql;
   connect to remote (server=srv1 dbms=db2 dbmsarg=(ssid=db25));
   select * from connection to remote
       (select * from db2table);
   disconnect from remote;
   quit;
   ```

This example uses a previously created view descriptor. The client has its own authority for the connections. The PreLib libref PreLib that was previously assigned and the client-assigned libref MyLib have no relevant difference. These are Base SAS engine librefs and not DB2 engine librefs.

1. In the server session
   ```sas
   libname prelib V8 'SAS.library';
   proc server id=srv1;
   run;
   ```

2. In the client session
   ```sas
   libname prelib server=srv1;
   proc print data=prelib.accview;
   run;
   ```

3. In the client session
libname mylib 'SAS.library2' server=srv1 rengine=v8;
proc print data=mylib.accview;
run;

**Known Issues with RRSAF Support**

SAS/SHARE can use various communication access methods to communicate with clients. You can specify these through the COMAMID and COMAUX1 system options.

When you use XMS (Cross Memory Services) as an access method, DB2 also uses XMS in the same address space. Predefining DB2 server librefs before starting PROC SERVER can result in errors due to the loss of the XMS Authorization Index, because both SAS and DB2 are acquiring and releasing it. When using XMS as an access method, use only client-assigned librefs on the server.

This problem does not occur when you use the TCP/IP access method. So if you use TCP/IP instead of XMS, you can use both client-assigned (client authority) and server-preassigned (server authority) librefs. You can also use either access method if your connection is not based on a libref (client authority).

---

**DB2 under z/OS Information for the Database Administrator**

**How the Interface to DB2 Works**

SAS/ACCESS Interface to DB2 under z/OS uses either the Call Attachment Facility (CAF) or the Recoverable Resource Management Services Attachment Facility (RRSAF) to communicate with the local DB2 subsystem. Both attachment facilities enable programs to connect to DB2 and to use DB2 for SQL statements and commands. SAS/ACCESS Interface to DB2 under z/OS uses the attachment facilities to establish and control its connections to the local DB2 subsystem. DB2 allows only one connection for each task control block (TCB), or task. SAS and SAS executables run under one TCB, or task.

The DB2 LIBNAME statement lets SAS users connect to DB2 more than once. Because the CAF and RRSAF allow only one connection per TCB, SAS/ACCESS Interface to DB2 under z/OS attaches a subtask for each subsequent connection that is initiated. It uses the ATTACH, DETACH, POST, and WAIT assembler macros to create and communicate with the subtasks. It does not limit the number of connections or subtasks that a single SAS user can initiate. This image illustrates how the DB2 engine works.
How and When Connections Are Made

SAS/ACCESS Interface to DB2 under z/OS always makes an explicit connection to the local DB2 subsystem (SSID). When a connection executes successfully, a thread to DB2 is established. For each thread’s or task’s connection, DB2 establishes authorization identifiers (AUTHIDs).

The DB2 interface determines when to make a connection to DB2 based on the type of Open mode that a SAS application requests for the DB2 tables. The Open mode can be Read, Update, or Output. Here is the default behavior.

- SAS/ACCESS Interface to DB2 under z/OS shares the connection for all openings in Read mode for each DB2 LIBNAME statement
- SAS/ACCESS Interface to DB2 under z/OS acquires a separate connection to DB2 for every opening in Update or Output mode.

You can change this default behavior by using the CONNECTION= option.

Several SAS applications require SAS/ACCESS Interface to DB2 under z/OS to query the DB2 system catalogs. When this type of query is required, the DB2 interface acquires a separate connection to DB2 to avoid contention with other applications that are accessing the DB2 system catalogs. See “Accessing DB2 System Catalogs” on page 581 for more information.

The DEFER= LIBNAME option also controls when a connection is established. UTILCONN_TRANSIENT= also allows control of the utility connection—namely, whether it must stay open.

DDF Communication Database

DB2 Distributed Data Facility (DDF) Communication Database (CDB) enables DB2 z/OS applications to access data on other systems. Database administrators are responsible for customizing CDB. SAS/ACCESS Interface to DB2 under z/OS supports both types of DDF: system-directed access (private protocol) and Distributed Relational Database Architecture.

*System-directed access* enables one DB2 z/OS subsystem to execute SQL statements on another DB2 z/OS subsystem. System-directed access uses a private protocol only for DB2. It is known as a private protocol because you can use only it between DB2 databases. IBM recommends that users use DRDA. Although SAS/ACCESS Interface to
DB2 under z/OS cannot explicitly request a connection, it can instead perform an implicit connection when SAS initiates a distributed request. To initiate an implicit connection, you must specify the LOCATION= option. When you specify this option, the three-level table name (location.authid.table) is used in the SQL statement that SAS/ACCESS Interface to DB2 under z/OS generates. When the SQL statement that contains the three-level table name is executed, an implicit connection is made to the remote DB2 subsystem. The primary authorization ID of the initiating process must be authorized to connect to the remote location.

Distributed Relational Database Architecture (DRDA) is a set of protocols that lets a user access distributed data. This lets SAS/ACCESS Interface to DB2 under z/OS access multiple remote tables at various locations. The tables can be distributed among multiple platforms, and both like and unlike platforms can communicate with one another. In a DRDA environment, DB2 acts as the client, server, or both.

To connect to a DRDA remote server or location, SAS/ACCESS Interface to DB2 under z/OS uses an explicit connection. To establish an explicit connection, SAS/ACCESS Interface to DB2 under z/OS first connects to the local DB2 subsystem through an attachment facility (CAF or RRSAF). It then issues an SQL CONNECT statement to connect from the local DB2 subsystem to the remote DRDA server before it accesses data. To initiate a connection to a DRDA remote server, you must specify the SERVER= connection option. When you specify this option, SAS uses a separate connection for each remote DRDA location.

**DB2 Attachment Facilities (CAF and RRSAF)**

By default, SAS/ACCESS Interface to DB2 under z/OS uses the Call Attachment Facility (CAF) to make its connections to DB2. SAS supports multiple CAF connections for a SAS session. Therefore, for a SAS server, all clients can have their own connections to DB2; multiple clients no longer have to share one connection. However, because CAF does not support sign-on, each connection that the SAS server makes to DB2 has the z/OS authorization identifier of the server. It does not have the authorization identifier of the client for which the connection is made.

If you specify the DB2ERRS system option, SAS/ACCESS Interface to DB2 under z/OS engine uses the Recoverable Resource Manager Services Attachment Facility (RRSAF). Only one attachment facility can be used at a time, so the DB2RRS or NODB2RRS system option can be specified only when a SAS session is started. SAS supports multiple RRSAF connections for a SAS session. RRSAF is a new feature in DB2 Version 5, Release 1, and its support in SAS/ACCESS Interface to DB2 under z/OS was new in SAS 8.

The RRSAF is intended for use by SAS servers, such as the ones that SAS/SHARE software use. RRSAF supports the ability to associate a z/OS authorization identifier with each connection at sign-on. This authorization identifier is not the same as the authorization ID that you specify in the AUTHID= data set or LIBNAME option. When connections use the System Authorization Facility and other security products, such as RACF, DB2 uses the RRSAF-supported authorization identifier to validate a given connection's authorization to use both DB2 and system resources. This authorization identifier is basically the user ID with which you are logged on to z/OS.

With RRSAF, the SAS server makes the connections for each client. These connections are associated with the client z/OS authorization identifier. This is true only for clients that the SAS server authenticated, which occurred when the client specified a user ID and password. Servers authenticate their clients when the clients provide their user IDs and passwords. Generally, this is the default way that servers are run. If a client connects to a SAS server without providing a user ID and password, the identifier associated with its connections is that of the server (as with CAF) and not the client’s identifier.
Other than specifying DB2RRS at SAS start-up, you do not need to do anything else to use RSSAF. The DB2 interface automatically signs on each connection that it makes to DB2 with the identifier of either the authenticated client or of the SAS server for non-authenticated clients. Authenticated clients have the same authorities to DB2 as they have when they run their own SAS session from their own ID and access DB2.

**Accessing DB2 System Catalogs**

For many types of SAS procedures, the DB2 interface must access DB2 system catalogs for information. This information is limited to a list of all tables for a specific authorization identifier. The interface generates this SQL query to obtain information from system catalogs:

```sql
SELECT NAME FROM SYSIBM.SYSTABLES
  WHERE (CREATOR = 'authid');
```

Unless you specify the `AUTHID=` option, the authorization ID is the z/OS user ID that is associated with the job step.

The SAS procedures or applications that request the list of DB2 tables includes, but is not limited to, PROC DATASETS and PROC CONTENTS, or any application that needs a member list. If the SAS user does not have the necessary authorization to read the DB2 system catalogs, the procedure or application fails.

Because querying the DB2 system catalogs can cause some locking contentions, SAS/ACCESS Interface to DB2 under z/OS initiates a separate connection for the query to the DB2 system catalogs. After the query completes, a COMMIT WORK command is executed.

Under certain circumstances, you can access a catalog file by overriding the default value for the `DB2CATALOG=` system option.
Chapter 18
SAS/ACCESS Interface to Greenplum

Introduction to SAS/ACCESS Interface to Greenplum ............................................ 584
LIBNAME Statement Specifics for Greenplum ......................................................... 584
  Overview ............................................................................................................. 584
  Arguments .......................................................................................................... 584
  LIBNAME Statement Examples ......................................................................... 587
Data Set Options for Greenplum ............................................................................ 588
SQL Pass-Through Facility Specifics for Greenplum .............................................. 590
  Key Information .................................................................................................. 590
  CONNECT Statement Example ......................................................................... 590
  Special Catalog Queries .................................................................................... 591
Autopartitioning Scheme for Greenplum ................................................................. 592
  Overview ............................................................................................................. 592
  Autopartitioning Restrictions ........................................................................... 592
  Nullable Columns ............................................................................................... 592
  Using WHERE Clauses ....................................................................................... 592
  Using DBSLICEPARM= ..................................................................................... 592
  Using DBSLICE= ................................................................................................. 592
Passing SAS Functions to Greenplum ..................................................................... 594
Passing Joins to Greenplum ..................................................................................... 595
Bulk Loading for Greenplum .................................................................................. 595
  Overview ............................................................................................................. 595
  Using Protocols to Access External Tables ....................................................... 595
  Configuring the File Server .............................................................................. 596
  Stopping gpfdist ................................................................................................ 596
  Troubleshooting gpfdist .................................................................................... 597
  Using the file:// Protocol .................................................................................. 597
  Accessing Dynamic Data in Web Tables .......................................................... 597
  Data Set Options for Bulk Loading .................................................................. 597
  Examples ............................................................................................................ 598
Naming Conventions for Greenplum ..................................................................... 599
Data Types for Greenplum .................................................................................... 599
  Overview ............................................................................................................. 599
  Supported Greenplum Data Types .................................................................. 599
  Greenplum Null Values ..................................................................................... 600
  LIBNAME Statement Data Conversions ............................................................ 601
Introduction to SAS/ACCESS Interface to Greenplum

For available SAS/ACCESS features, see Greenplum supported features on page 79. For more information about Greenplum, see your Greenplum documentation.

LIBNAME Statement Specifics for Greenplum

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Greenplum supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Greenplum.

LIBNAME libref greenplm <connection-options> <LIBNAME-options>;

Arguments

libref
specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

greenplm
specifies the SAS/ACCESS engine name for the Greenplum interface.

connection-options
provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to the Greenplum database in two ways. Specify only one of these methods for each connection because they are mutually exclusive.

• SERVER=, DATABASE=, PORT=, USER=, PASSWORD=
• DSN=, USER=, PASSWORD=

Here is how these options are defined.

SERVER=<server-name>
specifies the Greenplum server name or the IP address of the server host. If the server name contains spaces or nonalphanumeric characters or if it is an IP address, you must enclose it in quotation marks.

DATABASE=<database-name>
specifies the Greenplum database that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: DB=

PORT=port
specifies the port number that is used to connect to the specified Greenplum database. If you do not specify a port, the default is 5432.
USER=<<'Greenplum-user-name'>>
specifies the Greenplum user name (also called the user ID) that is used to connect to the database. If the user name contains spaces or nonalphanumeric characters, use quotation marks.

PASSWORD=<<'Greenplum-password'>>
specifies the password that is associated with your Greenplum user ID. If the password contains spaces or nonalphabetic characters, you must enclose it in quotation marks. You can also specify PASSWORD= with the PWD=, PASS=, and PW= aliases.

DSN=<<'Greenplum-data-source'>>
specifies the configured Greenplum ODBC data source to which you want to connect. It is recommended that you use this option only if you have configured Greenplum ODBC data sources on your client. This method requires additional setup—either through the ODBC Administrator control panel on Windows platforms, or through the odbc.ini file or a similarly named configuration file on UNIX platforms. It is recommended that you use this connection method only if you have existing, functioning data sources that have been defined.

LIBNAME -options
define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Greenplum with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

Table 18.1 SAS/ACCESS LIBNAME Options for Greenplum

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>operation-specific</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>FORWARD_ONLY</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>DLBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DLBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>none</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>none</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARAM=</td>
<td>THREADED_APPS</td>
</tr>
<tr>
<td>DEFER=</td>
<td>none</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>none</td>
</tr>
<tr>
<td>IN=</td>
<td>none</td>
</tr>
<tr>
<td>INSERT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>KEYSET_SIZE=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for Greenplum”</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>see “Naming Conventions for Greenplum”</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>RC</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SQLGENERATION=</td>
<td>DBMS</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>none</td>
</tr>
<tr>
<td>TRACE=</td>
<td>none</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>RC</td>
</tr>
<tr>
<td>UPDATE_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>none</td>
</tr>
</tbody>
</table>

**LIBNAME Statement Examples**

In this example, SERVER=, DATABASE=, PORT=, USER=, and PASSWORD= are the connection options.

```plaintext
libname mydblib greenplm server=gplum04 db=customers port=5432
   user=gpusr1 password=gppwd1;

proc print data=mydblib.customers;
   where state='CA';
run;
```

In the next example, DSN=, USER=, and PASSWORD= are the connection options. The Greenplum data source is configured in the ODBC Administrator Control Panel on Windows platforms. It is also configured in the odbc.ini file or a similarly named configuration file on UNIX platforms.

```plaintext
libname mydblib greenplm DSN=gplumSalesDiv user=gpusr1 password=gppwd1;

proc print data=mydblib.customers;
   where state='CA';
run;
```
Data Set Options for Greenplum

All SAS/ACCESS data set options in this table are supported for Greenplum. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

Table 18.2  SAS/ACCESS Data Set Options for Greenplum

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_DATAFILE_EXISTS=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DELTE_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DELIMITER=</td>
<td></td>
</tr>
<tr>
<td>BL_ENCODING=</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>BL_ESCAPE=</td>
<td>\</td>
</tr>
<tr>
<td>BL_EXCEPTION=</td>
<td>none</td>
</tr>
<tr>
<td>BL_EXECUTE_CMD=</td>
<td>none</td>
</tr>
<tr>
<td>BL_EXECUTE_LOCATION=</td>
<td>none</td>
</tr>
<tr>
<td>BL_EXTERNAL_WEB=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_FORCE_NOT_NULL=</td>
<td>none</td>
</tr>
<tr>
<td>BL_FORMAT=</td>
<td>TEXT</td>
</tr>
<tr>
<td>BL_HEADER=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_HOST=</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>BL_NULL=</td>
<td>\N [TEXT mode], unquoted empty value [CSV mode]</td>
</tr>
<tr>
<td>BL_PORT=</td>
<td>8080</td>
</tr>
<tr>
<td>BL_PROTOCOL=</td>
<td>'gpfdist'</td>
</tr>
<tr>
<td>BL_QUOTE=</td>
<td>&quot; (double quotation mark)</td>
</tr>
<tr>
<td>BL_REJECT_LIMIT=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>BL_REJECT_TYPE=</td>
<td>ROWS</td>
</tr>
<tr>
<td>BL_USE_PIPE=</td>
<td></td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>none</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>none</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for Greenplum”</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for Greenplum”</td>
</tr>
<tr>
<td>DISTRIBUTED_BY=</td>
<td>DISTRIBUTEDRANDOMLY</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>none</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>

SQL Pass-Through Facility Specifics for Greenplum

Key Information

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Greenplum interface.

- The *dbms-name* is **GREENPLM**.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to Greenplum. If you use multiple simultaneous connections, you must use the *alias* argument to identify the different connections. If you do not specify an alias, the default **GREENPLM** alias is used.
- The CONNECT statement *database-connection-arguments* are identical to its LIBNAME connection options.

CONNECT Statement Example

This example uses the DBCON alias to connect to the **greenplum04** Greenplum server database and execute a query. The connection alias is optional.

```sql
proc sql;
  connect to greenplm as dbcon
  (server=greenplum04 db=sample port=5432 user=gpusr1 password=gppwd1);
```
select * from connection to dbcon
  (select * from customers where customer like '1%');
quit;

Special Catalog Queries

SAS/ACCESS Interface to Greenplum supports the following special queries. You can use the queries to call functions in ODBC-style function application programming interfaces (APIs). Here is the general format of the special queries:

Greenplum::SQLAPI 'parameter-1', 'parameter-n'

Greenplum:: is required to distinguish special queries from regular queries. Greenplum:: is not case sensitive.

SQLAPI is the specific API that is being called. SQLAPI is not case sensitive.

'parameter n' is a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters, and the underscore represents any single character. To use either character as a literal value, you can use the backslash character (\) to escape the match characters. For example, this call to SQLTables usually matches table names such as myatest and my_test:

select * from connection to greenplm
  (Greenplum::SQLTables "test","","my_test");

Use the escape character to search only for the my_test table:

select * from connection to greenplm
  (Greenplum::SQLTables "test","","my\_test");

SAS/ACCESS Interface to Greenplum supports these special queries.

Greenplum::SQLTables <'Catalog', 'Schema', 'Table-name', 'Type'>
  returns a list of all tables that match the specified arguments. If you do not specify any arguments, all accessible table names and information are returned.

Greenplum::SQLColumns <'Catalog', 'Schema', 'Table-name', 'Column-name'>
  returns a list of all tables that match the specified arguments. If you do not specify any arguments, all accessible column names and information are returned.

Greenplum::SQLPrimaryKeys <'Catalog', 'Schema', 'Table-name', 'Type'>
  returns a list of all columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If you do not specify any table name, this special query fails.

Greenplum::SQLStatistics <'Catalog', 'Schema', 'Table-name'>
  returns a list of the statistics for the specified table name. You can set the SQL_INDEX_ALL and SQL_ENSURE options in the SQLStatistics API call. If you do not specify any table name argument, this special query fails.
Autopartitioning Scheme for Greenplum

Overview

Autopartitioning for SAS/ACCESS Interface to Greenplum is a modulo (MOD) function method. For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

Autopartitioning Restrictions

SAS/ACCESS Interface to Greenplum places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here is how columns are partitioned.

- INTEGER and SMALLINT columns are given preference.
- You can use other numeric columns for partitioning if the precision minus the scale of the column is greater than 0 but less than 10—namely, 0 < (precision-scale) < 10.

Nullable Columns

If you select a nullable column for autopartitioning, the OR<column-name> IS NULL SQL statement is appended at the end of the SQL code that is generated for the threaded Read. This ensures that any possible NULL values are returned in the result set.

Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in a SAS WHERE clause. For example, this DATA step cannot use a threaded Read to retrieve the data because all numeric columns in the table are in the WHERE clause:

data work.locemp;
set trlib.MYEMPS;
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;

Using DBSLICEPARM=

Although SAS/ACCESS Interface to Greenplum defaults to three threads when you use autopartitioning, do not specify a maximum number of threads for the threaded Read in DBSLICEPARM= LIBNAME option on page 151.

Using DBSLICE=

You might achieve the best possible performance when using threaded Reads by specifying the DBSLICE= data set option for Greenplum in your SAS operation. This is
especially true if your Greenplum data is evenly distributed across multiple partitions in a Greenplum database system.

When you create a Greenplum table using the Greenplum database partition model, you can specify the partitioning key that you want to use by appending the `PARTITION BY <column-name>` clause to your `CREATE TABLE` statement. Here is how you can accomplish this by using the `DB_CREATE_TABLE_OPTS=LIBNAME` option within the SAS environment.

```sas
/* Points to a triple-node server. */
libname mylib greenplm server=gplum03 user=myuser pw=mypwd db=greenplum;

DB_CREATE_TABLE_OPTS='PARTITION BY(EMPNUM);

proc datasets library=mylib;
  delete MYEMPS1;run;

data mylib.myemps(drop=morf whatstate
  DBTYPE=(HIREDATE="date" SALARY="numeric(8,2)"
           NUMCLASS="smallint" GENDER="char(1)" ISTENURE="numeric(1)" STATE="char(2)"
           EMPNUM="int NOT NULL Primary Key");
  format HIREDATE mmddyy10.;
  do EMPNUM=1 to 100;
    morf=mod(EMPNUM,2)+1;
    if(morf eq 1) then
      GENDER='F';
    else
      GENDER='M';
    SALARY=(ranuni(0)*5000);
    HIREDATE=int(ranuni(13131)*3650);
    whatstate=int(EMPNUM/5);
    if(whatstate eq 1) then
      STATE='FL';
    if(whatstate eq 2) then
      STATE='GA';
    if(whatstate eq 3) then
      STATE='SC';
    if(whatstate eq 4) then
      STATE='VA';
    else
      state='NC';
    ISTENURE=mod(EMPNUM,2);
    NUMCLASS=int(EMPNUM/5)+2;
    output;
  end;
  run;

After the MYEMPS table is created on this three-node database, a third of the rows reside on each of the three nodes.

Using `DBSLICE=` works well when the table that you want to read is not stored in multiple partitions. It gives you flexibility in column selection. For example, if you know that the `STATE` column in your employee table contains only a few distinct values, you can modify your `DBSLICE=` option accordingly.

```sas
data work.locemp;
  set mylib.MYEMPS (DBSLICE=('STATE='GA'
                      'STATE='SC'
                      'STATE='VA'
                      'STATE='NC'));
  where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
```
Passing SAS Functions to Greenplum

SAS/ACCESS Interface to Greenplum passes the following SAS functions to Greenplum for processing. Where the Greenplum function name differs from the SAS function name, the Greenplum name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

** (POWER(base, exponent))  LOWCASE (LOWER)
ABS                  MAX
ARCOS (ACOS)         MIN
AR Sin (ASIN)        MINUTE (DATEPART)
ATAN                MONTH (DATEPART)
ATAN2              QTR (DATEPART)
AVG                  REPEAT
BYTE (CHR)           SECOND (DATEPART)
CEIL                SIGN
COMPRESS (TRANSLATE) SIN
COS                  SQRT
COT                  STRIP (BTRIM)
COUNT               SUBSTR (SUBSTRING)
DAY (DATEPART)       SUM
EXP                  TAN
FLOOR               TRANWRD (REPLACE)
HOUR (DATEPART)      TRIMN (RTRIM)
INDEX (STRPOS)      UPCASE (UPPER)
LENGTH            WEEKDAY (DATEPART)
LOG (LN)          YEAR (DATEPART)
LOG10 (LOG)

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Greenplum. Due to incompatibility in date and time functions between Greenplum and SAS, Greenplum might not process them correctly. Check your results to determine whether these functions are working as expected.

DATE (NOW)            TIME (current_time)
DATEPART (CONVERT)    TIMEPART (TIME)
DATETIME (NOW)        TODAY (NOW)
Passing Joins to Greenplum

For a multiple libref join to pass to Greenplum, all of these components of the LIBNAME statements must match exactly.

- user ID (USER=)
- password (PASSWORD=)
- host (HOST=)
- server (SERVER=)
- database (DATABASE=)
- port (PORT=)
- data source (DSN=, if specified)
- SQL functions (SQL_FUNCTIONS=)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for Greenplum

Overview

Bulk loading provides high-performance access to external data sources. Multiple Greenplum instances read data in parallel, which enhances performance.

Bulk loading is the fastest way to insert large numbers of rows into Greenplum tables. You can also use bulk loading to execute high-performance SQL queries against external data sources, without first loading those data sources into a Greenplum database. These fast SQL queries let you optimize extraction, transformation, and loading tasks that are common in data warehousing.

Two types of external data sources, external tables and web tables, have different access methods. External tables contain static data that can be scanned multiple times. The data does not change during queries. Web tables provide access to dynamic data sources as if those sources were regular database tables. Web tables cannot be scanned multiple times. The data can change during the course of a query.

You must specify BULKLOAD=YES to use the bulk-load facility.

The following sections show you how to access external tables and web tables using the bulk-load facility.

Using Protocols to Access External Tables

Use these protocols to access (static) external tables.

gpfdist://

To use the gpfdist:// protocol, install and configure the gpfdist (Greenplum file distribution) program on the host that stores the external tables, see “Configuring the File Server”. The gpfdist utility serves external tables in parallel to the primary
Greenplum database segments. The gpfdist:// protocol is advantageous because it ensures that all Greenplum database segments are used during the loading of external tables.

To specify files to gpfdist, use the BL_DATAFILE= data set option. Specify file paths that are relative to the directory from which gpfdist is serving files (the directory where you executed gpfdist).

The gpfdist utility is part of the loader package for the platform where SAS is running. You can also download it from the Pivotal website: http://pivotal.io/.

file://

To use the file:// protocol, external tables must reside on a segment host in a location that Greenplum superusers (gpadmin) can access. The segment host name must match the host name, as specified in the gp_configuration system catalog table. In other words, the external tables that you want to load must reside on a host that is part of the set of servers that comprise the database configuration. The file:// protocol is advantageous because it does not require configuration.

## Configuring the File Server

Follow these steps to configure the gpfdist file server.

2. Define and load a new environment variable called GPLOAD_HOME.
3. Set the value of the variable to the directory that contains the external tables that you want to load.
   The directory path must be relative to the directory in which you execute gpfdist, and it must exist before gpfdist tries to access it.
   - For Windows, open My Computer, select the Advanced tab, and click the Environment Variables button.
   - For UNIX, enter this command or add it to your profile:
     ```
     export GPLOAD_HOME=directory
     ```
4. Start gpfdist as shown in these examples.
   - For Windows:
     ```
     C:> gpfdist -d %GPLOAD_HOME% -p 8080 -l %GPLOAD_HOME%/gpfdist.log
     ```
   - For UNIX:
     ```
     $ gpfdist -d $GPLOAD_HOME -p 8080 -l $GPLOAD_HOME/gpfdist.log &
     ```

You can run multiple instances of gpfdist on the same host as long each instance has a unique port and directory.

If you do not set GPLOAD_HOME, the value of the BL_DATAFILE= data set option specifies the directory that contains the external tables to be loaded. If BL_DATAFILE is not specified, the current directory is assumed to contain the external tables.

## Stopping gpfdist

In Windows, to stop an instance of gpfdist, use the Task Manager or close the Command Window that you used to start that instance of gpfdist.

Follow these steps in UNIX to stop an instance of gpfdist.
1. Find the process ID:

   $ ps ax | grep gpfdist (Linux)
   $ ps -ef | grep gpfdist (Solaris)

2. Kill the process. Here is an example:

   $ kill 3456

**Troubleshooting gpfdist**

Run this command to test connectivity between an instance of gpfdist and a Greenplum database segment.

   $ wget http://gpfdist_hostname:port/filename

**Using the file:// Protocol**

You can use the file:// protocol to identify external files for bulk loading with no additional configuration required. However, using the GPLOAD_HOME environment variable is highly recommended. If you do not specify GPLOAD_HOME, the BL_DATAFILE data set option specifies the source directory. The default source directory is the current directory if you do not set BL_DATAFILE=. The Greenplum server must have access to the source directory.

**Accessing Dynamic Data in Web Tables**

Use these data set options to access web tables:

- **BL_EXECUTE_CMD**=
- **BL_LOCATION**=

**Data Set Options for Bulk Loading**

Here are the Greenplum bulk-load data set options. For detailed information about these options, see Data Set Options for Relational Databases on page 241.

- **BL_DATAFILE**=
- **BL_CLIENT_DATAFILE**=
- **BL_DELETE_DATAFILE**=
- **BL_DELIMITER**=
- **BL_ENCODING**=
- **BL_ESCAPE**=
- **BL_EXCEPTION**=
- **BL_EXECUTE_CMD**=
- **BL_EXECUTE_LOCATION**=
- **BL_EXTERNAL_WEB**=
- **BL_FORCE_NOT_NULL**=
- **BL_FORMAT**=
Examples

This first example shows how you can use a SAS data set, SASFLT.FLT98, to create and load a Greenplum table, FLIGHTS98.

```
libname sasflt 'SAS-data-library';
libname mydblib greenplm server=mysrv1_users
db=users user=myusr1 password=mypwd1;
proc sql;
create table net_air.flights98
  (BULKLOAD=YES
   BL_DATAFILE='c:\temp\greenplum\data.dat'
   BL_USE_PIPE=NO
   BL_DELETE_DATAFILE=yes
   BL_HOST='192.168.x.x'
   BL_PORT=8081)
  as select * from sasflt.flt98;
quit;
```

This next example shows how you can append the SAS data set, SASFLT.FLT98, to the existing Greenplum table ALLFLIGHTS. The BL_USE_PIPE=NO option forces SAS/ACCESS Interface to Greenplum to write data to a flat file, as specified in the BL_DATAFILE= option. Rather than deleting the data file, BL_DELETE_DATAFILE=NO causes the engine to leave it after the load has completed.

```
proc append base=new_air.flights98
  (BULKLOAD=YES
   BL_DATAFILE='c:\temp\greenplum\data.dat'
   BL_USE_PIPE=NO
   BL_DELETE_DATAFILE=no
   BL_HOST='192.168.x.x'
   BL_PORT=8081)
data=sasflt.flt98;
run;
```
Naming Conventions for Greenplum

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The Greenplum interface supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name results in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

The `PRESERVE_TAB_NAMES=` and `PRESERVE_COL_NAMES=` options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see `LIBNAME Statement for Relational Databases` on page 96.) Greenplum is not case sensitive, so all names default to lowercase.

Greenplum objects include tables, views, and columns. They follow these naming conventions.

- A name can contain up to 128 characters.
- The first character in a name can be a letter, @, _, or #.
- A name cannot be a Greenplum reserved word, such as WHERE or VIEW.
- A name must be unique within each type of each object.

Data Types for Greenplum

Overview

Every column in a table has a name and a data type. The data type tells Greenplum how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Greenplum data types, null and default values, and data conversions.

SAS/ACCESS Interface to Greenplum does not directly support any data types that are not listed below. Any columns using these types are read into SAS as character strings.

For more information about Greenplum data types and to determine which data types are available for your version of Greenplum, see your Greenplum documentation.

Supported Greenplum Data Types

Here are the data types that SAS/ACCESS Interface to Greenplum supports:

- Character data:
  - `CHAR(n)`
  - `VARCHAR(n)`
  - `TEXT`
• Numeric data:
  
  BIGINT   REAL
  SMALLINT FLOAT
  INTEGER  DECIMAL | DEC | NUMERIC
  DOUBLE PRECISION

  Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

• Date, time, and timestamp data:
  
  DATE
  TIME
  TIMESTAMP

  Note: Be aware that columns of these data types can contain data values that are out of range for SAS.

• Binary data: BYTEA

**Greenplum Null Values**

Greenplum has a special value called NULL. A Greenplum NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Greenplum NULL value, it interprets it as a SAS missing value. When loading SAS tables from Greenplum sources, SAS/ACCESS stores Greenplum NULL values as SAS missing values.

In Greenplum tables, NULL values are valid in all columns by default. There are two methods to define a column in a Greenplum table so that it requires data:

• Using SQL, you specify a column as NOT NULL. This tells SQL to allow only a row to be added to a table if a value exists for the field. Rows that contain NULL values in that column are not added to the table.

• Another approach is to assert NOT NULL DEFAULT.

When creating Greenplum tables with SAS/ACCESS, you can use the DBNULL= data set option to specify the treatment of NULL values. For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

Once you know whether a Greenplum column enables NULLs or the host system supplies a default value for a column that is defined as NOT NULL WITH DEFAULT, you can write selection criteria and enter values to update a table. Unless a column is defined as NOT NULL or NOT NULL DEFAULT, it enables NULL values.

To control how the DBMS handles SAS missing character values, use the NULLCHAR= and NULLCHARVAL= data set options.
**LIBNAME Statement Data Conversions**

The following table shows the default formats that SAS/ACCESS Interface to Greenplum assigns to SAS variables when using the `LIBNAME` statement to read from a Greenplum table.

These default formats are based on Greenplum column attributes.

**Table 18.3  LIBNAME Statement: Default SAS Formats for Greenplum Data Types**

<table>
<thead>
<tr>
<th>Greenplum Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>VARCHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>BYTEA</td>
<td>character</td>
<td>$w.***</td>
</tr>
<tr>
<td>INTEGER</td>
<td>numeric</td>
<td>11.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.</td>
</tr>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>DECIMAL(p,s)**</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>NUMERIC(p,s)**</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME8.</td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME25.6</td>
</tr>
</tbody>
</table>

* * *  

n in Greenplum data types is equivalent to w in SAS formats.  
** p and s in Greenplum numeric data types are equivalent to w and d in SAS formats.  
*** Because the Greenplum ODBC driver does the conversion, this field is displayed as if the $HEXw. format were applied.

The next table shows the default Greenplum data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the `LIBNAME` statement.

**Table 18.4  LIBNAME Statement: Default Greenplum Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Greenplum Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.d</td>
<td>DECIMAL(p,s)**</td>
</tr>
<tr>
<td>$w.</td>
<td>VARCHAR(n)*</td>
</tr>
<tr>
<td>SAS Variable Format</td>
<td>Greenplum Data Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>datetime formats</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>TIME</td>
</tr>
</tbody>
</table>

* n in Greenplum data types is equivalent to w in SAS formats.

** p and s in Greenplum numeric data types are equivalent to w and d in SAS formats.
Chapter 19
SAS/ACCESS Interface to Hadoop

Introduction to SAS/ACCESS Interface to Hadoop ........................................... 604
  Overview ........................................................................................................ 604
  Hadoop Concepts ......................................................................................... 604
  Hadoop Configuration ................................................................................. 605
  National Language Support Limitations ...................................................... 605

LIBNAME Statement Specifics for Hadoop ....................................................... 605
  Overview ........................................................................................................ 605
  JDBC Read Security .................................................................................... 606
  HDFS Write Security .................................................................................. 606
  HDFS Permission Requirements for Optimized Reads ................................. 606
  Arguments .................................................................................................... 606
  Hadoop LIBNAME Statement Examples ....................................................... 610

Data Set Options for Hadoop ........................................................................ 610

SQL Pass-Through Facility Specifics for Hadoop ........................................... 611
  Key Information ............................................................................................ 611
  CONNECT Statement Examples ................................................................... 612

Passing SAS Functions to Hadoop ................................................................. 612

Passing Joins to Hadoop ............................................................................... 613

Bulk Loading for Hadoop ............................................................................. 613
  Loading ......................................................................................................... 613
  Examples ...................................................................................................... 614

Naming Conventions for SAS and Hive ........................................................ 615

Data Types for Hadoop .................................................................................. 615
  Overview ...................................................................................................... 615
  Supported Hive Data Types ....................................................................... 616
  SAS Data Types .......................................................................................... 617
  Data Conversion from Hive to SAS ............................................................. 617
  Highlighting New Hive .12 Data Types ....................................................... 618
  Highlighting Support for Hive .13 Data Types ........................................... 618
  Issues When Converting Data from Hive to SAS ....................................... 618
  SAS Table Properties for Hive and Hadoop ............................................... 619
  Data Conversion from SAS to Hive ............................................................. 619
  Leverage Table Properties for Existing Hive Tables ................................... 620
  Address Issues When Converting Data from Hive to SAS .......................... 622
    with Table Properties .............................................................................. 622
Introduction to SAS/ACCESS Interface to Hadoop

Overview

For available SAS/ACCESS features, see Hadoop supported features on page 80. For more information about Hadoop, see your Hadoop documentation.

Hadoop Concepts

Hadoop is a general-purpose data storage and computing platform that includes database-like tools, such as Hive and HiveServer2. SAS/ACCESS Interface to Hadoop lets you work with your data using SQL constructs through Hive and HiveServer2. It also lets you access data directly from the underlying data storage layer, the Hadoop Distributed File System (HDFS). This differs from the traditional SAS/ACCESS engine behavior, which exclusively uses database SQL to read and write data.

With SAS/ACCESS Interface to Hadoop for Hive and Hive Server2, you can read and write data to and from Hadoop as if it were any other relational data source to which SAS can connect. This interface provides fast, efficient access to data stored in Hadoop through HiveQL. You can access Hive tables as if they were native SAS data sets and then analyze them using SAS.

Note: Support for HiveServer2 was added in the second maintenance release for SAS 9.4.

For Hive operations like submitting HiveQL, the Hadoop engine requires access to the Hive service that runs on the Hadoop cluster, often port 10000. For HDFS operations, such as writing data to Hive tables, the Hadoop engine requires access to the HDFS service that runs on the Hadoop cluster, often port 8020. If the Hadoop engine cannot access the HDFS service, its full functionality is not available.

Without the need for Hive and HiveServer2, the Hadoop engine can look in an assigned HDFS directory for metadata that SAS creates: files with a SASHDMD file type. Specify the HDFS_METADIR= option if you are working metadata that SAS creates. The Hadoop engine can create SASHDMD metadata when it writes output from SAS, or the HDMD procedure can create these metadata files.

For details about using the HDMD procedure to make your HDFS files available to the SAS system, see the HDMD Procedure in the Base SAS Procedures Guide.
Hadoop Configuration

Your Hadoop administrator configures the Hadoop cluster that you use. Your administrator has set the defaults for system parameters such as block size and replication factor that affect the Read and Write performance of your system. Replication factors greater than 1 help prevent data loss yet slow the writing of data. Consult with your administrator about how your particular cluster was configured.

To connect to a Hadoop server, you must complete these configuration steps.

• You must use a supported Hadoop distribution and configure a required set of Hadoop JAR files. The JAR files must be located in one location and available to the SAS client machine. The SAS environment variable SAS_HADOOP_JAR_PATH must be defined and point to the folders that contain all the Hadoop JAR files.

• Hadoop cluster configuration files include core-site.xml, hdfs-site.xml, hive-site.xml, mapred-site.xml, and, if applicable, yarn-site.xml. You must copy Hadoop configuration files from the Hadoop cluster to a physical location that the SAS client machine can access. You must also define and set the SAS environment variable SAS_HADOOP_CONFIG_PATH to the location of the Hadoop configuration files. If you are using MapR, no hdfs-site.xml file is required in the directory.

• To connect to the Hadoop server through WebHDFS, the SAS environment variable SAS_HADOOP_RESTFUL must be defined and set to the value 1. In addition, the Hadoop hdfs-site.xml configuration file must include the properties for the WebHDFS location.

Note: Support for the SAS_HADOOP_CONFIG_PATH and SAS_HADOOP_RESTFUL environment variables was added in the second maintenance release for SAS 9.4.

For more information, see the SAS 9.4 Hadoop Configuration Guide for Base SAS and SAS/ACCESS.

National Language Support Limitations

Keep these limitations in mind when working with Hadoop and Hive.

• Column names must use only WLATIN1 characters

• Table comments should also contain only WLATIN1 characters when the DBCREATE_TABLE_OPTS= data set option creates them using DBCREATE_TABLE_OPTS="COMMENT 'my table comment'". Although this option accepts any NLS character, the NLS portion of the comment is not displayed properly later.

LIBNAME Statement Specifics for Hadoop

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Hadoop supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Hadoop.
LIBNAME libref Hadoop <connection-options> <LIBNAME-options>

**JDBC Read Security**

SAS/ACCESS can access Hadoop data through a JDBC connection to a HiveServer or HiveServer2 service. Depending on what release of Hive you have, Hive might not implement Read security. If Hive does not implement Read security, a successful connection from SAS allows Read access to all data that is accessible to the Hive service. SAS/ACCESS can connect to a Hive or HiveServer2 service that is unsecured, user name and password secured, or secured by Kerberos. SAS/ACCESS supports Kerberos 5 Version 1.9 or later.

**HDFS Write Security**

SAS/ACCESS creates and appends to Hive tables using the HDFS service. SAS/ACCESS can connect to a Hive or HiveServer2 service that is unsecured, user name and password secured, or secured by Kerberos. Your HDFS connection needs Write access to the HDFS /tmp directory. After data is written to /tmp, a Hive LOAD command is issued on your JDBC connection to associate the data with a Hive table. Therefore, the JDBC Hive session also needs Write access to /tmp.

Note: If HDFS /tmp has enabled the sticky bit, the LOAD command can fail. To resolve this, either disable the /tmp sticky bit or use the HDFS_TEMPDIR option to specify an alternative HDFS directory for SAS/ACCESS to write data to.

**HDFS Permission Requirements for Optimized Reads**

To optimize big data reads, SAS/ACCESS creates a temporary table in HDFS /tmp. This requires that the SAS JDBC connection have Write access to /tmp. The temporary table is read using HDFS, so the SAS HDFS connection needs Read access to the temporary table that is written to /tmp. Alternatively, use the HDFS_TEMPDIR option to specify an HDFS directory to use instead of /tmp.

**Arguments**

*libref*

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

*Hadoop*

specifies the SAS/ACCESS engine name for the Hadoop interface.

*connection-options*

provide connection information and control how SAS manages the timing and concurrence of the connection to Hadoop. Here is how these options are defined.

Note: All of the following connection options are also valid in the CONNECT statement when you use the SQL pass-through facility (SQL procedure) to connect to your DBMS.

*USER=<Hadoop-user-name>*

specifies the user name for Read (JDBC) and Write (HDFS) operations. Do not use the USER= argument if your Hadoop cluster is secured by Kerberos.

Alias: UID=
PASSWORD='Hadoop-password'
    specifies the Hadoop password that is associated with your user ID. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you do not want to enter your Hadoop password in uncoded text on this statement, see PROC PWENCOD in the Base SAS Procedures Guide for a method to encode it.
    Alias: PASS=, PWD=, PW=

SERVER='Hadoop-server-name'
    specifies the Hadoop server name that runs the Hive service. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.
    Alias: HOST=

PORT=port
    specifies the port number that is used to connect to the specified Hive service.
    Alias: SERVICE=
    Default: 10000

SCHEMA='Hive-schema'
    specifies the Hive schema.
    Alias: DATABASE=, DB=
    Default: default

CFG='config-file'
    specifies a Hadoop configuration file for SAS/ACCESS to use. This file contains entries for Hadoop system information, including file system properties such as fs.defaultFS. The configuration file can be a copy of the Hadoop core-site.xml file. However, if your Hadoop cluster is running with HDFS failover enabled, you need to create a file that combines the contents of the Hadoop core-site.xml and hdfs-site.xml, if it exists. If you are using MapR, no hdfs-site.xml file is required in the directory.

    In the second maintenance release for SAS 9.4, the CFG= option is deprecated. Instead of specifying the CFG= option, you can define the SAS environment variable SAS_HADOOP_CONFIG_PATH to set the location of the Hadoop cluster configuration files: core-site.xml, hdfs-site.xml, hive-site.xml, mapred-site.xml, and, if applicable, yarn-site.xml. For instructions on making the required Hadoop cluster configuration files available to your SAS machine by using the SAS_HADOOP_CONFIG_PATH environment variable, see SAS 9.4 Hadoop Configuration Guide for Base SAS and SAS/ACCESS.

    Note: If you are using MapR, no hdfs-site.xml file is required in the directory.
    Alias: CONFIG=

HDFS_DATADIR=path'
    when not in Hive mode, specifies the path to the Hadoop directory where SAS reads and writes data (for example, '/sas/hpa'). Use this option only when you are not using Hive or HiveServer2. For details, see the “Accessing Data Independently from Hive” topic in the Base SAS Procedures Guide

    Note: Support for this connection option was added for SAS 9.4.
    Alias: HDFS_PERMDIR=
HDFS_METADIR='path'
specifies the path to an HDFS directory that contains XML-based table
definitions, called SASHMD descriptors. Through these descriptors, SAS then
accesses the data using HDFS instead of Hive. If you want the Hadoop engine to
close to Hive and use HiveQL, do not set this option.

*Note:* Support for this connection option was added for SAS 9.4.

HDFS_PRINCIPAL=
specifies the HDFS principal string in an environment that uses Kerberos.

Default: none

This argument is not required if the core-site.xml and hdfs-site.xml files exist in
the directory that is specified by the SAS_HADOOP_CONFIG_PATH
environment variable. If you are using MapR, no hdfs-site.xml file is required in
the directory.

*Note:* Support for HDFS_PRINCIPAL= was added for SAS 9.4. Support for the
SAS_HADOOP_CONFIG_PATH environment variable was added in the
second maintenance release for SAS 9.4.

HDFS_TEMPDIR='path'
specifies the path to the HDFS directory where SAS reads and writes temporary
data.

Default: HDFS_TEMPDIR='/tmp'

*Note:* Support for this connection option was added for SAS 9.4.

HIVE_PRINCIPAL=
specifies the Hive principal string in an environment that uses Kerberos.

Default: none

This argument is not required if hive-site.xml exists in the directory that is
specified by the SAS_HADOOP_CONFIG_PATH environment variable.

Alias: HIVE_KERBEROS_PRINCIPAL=

*Note:* Support for this connection option was added for SAS 9.4.

SUBPROTOCOL=Hive | Hive2
specifies whether you are connecting to HiveServer or HiveServer2. HiveServer2
is recommended if your Hadoop distribution supports it. If you do not specify a
value for the SUBPROTOCOL= connection option, the system first attempts to
connect to HiveServer2. If the connection fails, SAS notifies you in the SAS log
and then attempts to connect to HiveServer.

Alias: SUBPROTO=

Default: Hive2

*Note:* This option will be removed in a future release.

*Note:* Support for the SUBPROTOCOL= connection option was added for SAS
9.4. Support for HiveServer2 was added in the second maintenance release
for SAS 9.4. In addition, the default for SUBPROTOCOL= became Hive2 in
the second maintenance release for SAS 9.4.

**LIBNAME-options**
define how SAS processes DBMS objects. The following table describes the
LIBNAME options for SAS/ACCESS Interface to Hadoop, with the applicable
default values. This table also identifies LIBNAME options that are valid in the
CONNECT statement in the SQL procedure. For details, see LIBNAME Options for Relational Databases on page 101.

**Table 19.1  SAS/ACCESS LIBNAME Options for Hadoop**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
<th>Valid in CONNECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>BL_PORT=</td>
<td>8020</td>
<td></td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>CONFIG=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SAS_HADOOPCONFIG_PATH is the default when you set only BULKLOAD=YES)</td>
<td></td>
</tr>
<tr>
<td>CONFIGDIR=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SAS_HADOOP_JAR_PATH is the default when you set only BULKLOAD=YES)</td>
<td></td>
</tr>
<tr>
<td>DBCREATE_TABLE_EXTERNAL=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
<td></td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
<td></td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>LOGIN_TIMEOUT=</td>
<td>0</td>
<td>●</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPTS=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>PROPERTIES=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>READ_METHOD=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
<td>●</td>
</tr>
</tbody>
</table>
### Hadoop LIBNAME Statement Examples

This example uses the default HiveServer2 port and schema.

```plaintext
libname hdp sasiohdp server=hxpeduped
   user=myusr1 password=mypwd1;
```

This example explicitly specifies the default HiveServer2 port and schema.

```plaintext
libname hdp sasiohdp server=hxpeduped port=10000 schema=default
   user=myusr1 password=mypwd1;
```

This example specifies the Hive Kerberos principal to connect to a Kerberos secured HiveServer2 instance. It assumes that a Kerberos `kinit` has been successfully performed.

```plaintext
libname hdp sasiohdp server=hxpeduped
   hive_principal='hive/_HOST@HD.COMPANY.COM';
```

This example assumes a Kerberos secured HiveServer2 instance. It also assumes that SAS_HADOOP_CONFIG_PATH is set to a directory that contains the Hadoop hive-site.xml configuration file. Finally, this example assumes that a Kerberos `kinit` has been successfully performed.

```plaintext
libname hdp sasiohdp server=hxpeduped;
```

### Data Set Options for Hadoop

All SAS/ACCESS data set options in this table are supported for Hadoop. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

#### Table 19.2 SAS/ACCESS Data Set Options for Hadoop

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
<th>Valid in CONNECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_DELIMITER=</td>
<td>¶001 (Ctrl-A)</td>
<td></td>
</tr>
<tr>
<td>CONFIG=</td>
<td>LIBNAME option setting</td>
<td></td>
</tr>
<tr>
<td>CONFIGDIR=</td>
<td>LIBNAME option setting</td>
<td></td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DBCREATE_TABLE_EXTERNAL=</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for Hadoop

#### Key Information

For general information about this feature, see “Overview of SQL Procedure Interactions with SAS/ACCESS” on page 471.

Here are the SQL pass-through facility specifics for the Hadoop interface.

- The *dbms-name* is **HADOOP**.
- The CONNECT statement is required.

#### Option Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCREATE_TABLE_LOCATION</td>
<td>/user/hive/warehouse/<code>tabname</code> [with the default schema], /user/hive/warehouse/schema.db/<code>tabname</code> [with a nondefault schema]</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBGEN_NAME</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBLABEL</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT</td>
<td>none</td>
</tr>
<tr>
<td>DBSASLABEL</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE</td>
<td>see “Data Types for Hadoop” on page 615</td>
</tr>
<tr>
<td>DBTYPE</td>
<td>see “Data Types for Hadoop” on page 615</td>
</tr>
<tr>
<td>POST_STMT_OPTS</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS</td>
<td>none</td>
</tr>
<tr>
<td>READ_METHOD</td>
<td>none</td>
</tr>
<tr>
<td>ROW_DELIMITER</td>
<td>\010 (<code>\n</code>)</td>
</tr>
<tr>
<td>TRANSCODE_FAIL</td>
<td>ERROR</td>
</tr>
</tbody>
</table>
PROC SQL supports multiple connections to Hadoop. If you use multiple simultaneous connections, you must use the alias argument to identify the different connections. If you do not specify an alias, the default HADOOP alias is used.

The CONNECT statement database-connection-arguments are identical to its LIBNAME connection options.

**CONNECT Statement Examples**

This example uses the default HiveServer2 port and schema.

```sql
proc sql;
  connect to hadoop (user="myusr1" pw="mypwd1" server=hxpduped);
```

This example explicitly specifies the default HiveServer2 port and schema.

```sql
proc sql;
  connect to hadoop (user="myusr1" pw="mypwd1"
    server=hxpduped port=10000 schema=default);
```

This example specifies the Hive Kerberos principal to connect to a Kerberos secured HiveServer2 instance. It assumes that a Kerberos `kinit` has been successfully performed.

```sql
proc sql;
  connect to hadoop (hive_principal='hive/_HOST@HD.COMPANY.COM'
    server=hxpduped);
```

---

**Passing SAS Functions to Hadoop**

SAS/ACCESS Interface to Hadoop passes the following SAS functions to Hadoop for processing. Where the Hadoop function name differs from the SAS function name, the Hadoop name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

- ABS
- ARCOS (ACOS)
- ARSIN (ASIN)
- ATAN
- AVG
- CEIL
- COALESCE
- COMPRESS (REGEXP_REPLACE)*
- COS
- COUNT
- DAY
- EXP
- FLOOR
- HOUR
- INDEX (LOCATE)
- LENGTH
- LOWCASE (LOWER)
- MAX
- MIN
- MINUTE
- MONTH
- SECOND
- SIN
- SQRT
- STD (STDDEV_SAMP)
- STRIP (TRIM)
- SUBSTR
- SUM
- TAN
- TRANSTRN (REGEXP_REPLACE)
- TRIMN (RTRIM)
- UPCASE (UPPER)
LOG (LN)          VAR (VAR_SAMP)
LOG10             YEAR
* Only when you specify two arguments; for example, COMPRESS(string,' ').

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding Hadoop functions that are passed down to Hadoop. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Hadoop. Due to incompatibility in date and time functions between Hadoop and SAS, Hadoop might not process them correctly. Check your results to determine whether these functions are working as expected.

- DATE (TO_DATE(UNIX_TIMESTAMP))
- DATEPART (TO_DATE(UNIX_TIMESTAMP))
- DATETIME (FROM_UNIXTIME(UNIX_TIMESTAMP))
- REPEAT (TO_DATE(UNIX_TIMESTAMP))
- TODAY (TO_DATE(UNIX_TIMESTAMP))
- TRANSTRN (REGEXP_REPLACE)

### Passing Joins to Hadoop

The SAS/ACCESS engine does not pass LIBNAME-referenced cross-schema joins to Hadoop. To pass a multiple-libref join to Hadoop, the schemas for each LIBNAME statement must be identical. You can use the SQL pass-through facility to pass a cross-schema join to Hadoop.

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

### Bulk Loading for Hadoop

#### Loading

SAS/ACCESS Interface to Hadoop has no differentiation between bulk loading and a standard load process. Although BULKLOAD=YES syntax is supported, it does not change the underlying load process.

Here is how a text-based table (STORED AS TEXTFILE) is created.

1. SAS issues a CREATE TABLE Hive command to the Hive server. The command contains all table metadata (column definitions) and the table properties that are specific to SAS that refine Hive metadata to handle maximum string lengths and date/time formats.
2. SAS uses HDFS to upload table data to the HDFS /tmp directory. The resulting file is a UTF-8-delimited text file that by default uses CTRL-A (‘\001’) as a field delimiter and newline (‘\n’) as a record separator.
3. SAS issues a LOAD DATA command to move the data file from the HDFS /tmp directory to the appropriate Hive warehouse location. The data file is now part of the Hive table.

Here is how a non-text table is created:
1. Specify the DBCREATE_TABLE_OPTS= data set option containing a Hive 
STORED AS clause to the new table reference. Here is an example:

```plaintext
data hdp.new_hive_table(DBCREATE_TABLE_OPTS='STORED AS SEQUENCEFILE');
set sas_table;
run;
```

2. SAS issues two CREATE TABLE statements to the Hive server. One CREATE 
TABLE creates the target Hive table. The other CREATE TABLE creates a 
temporary table.

3. SAS uses HDFS to upload table data to the HDFS /tmp directory. The resulting file 
is a UTF-8-delimited text file.

4. SAS issues a LOAD DATA command to move the data file from the HDFS /tmp 
directory into the temporary table

5. SAS issues an INSERT INTO statement that copies and transforms the temp table 
text data into the target Hive table non-text format

6. SAS deletes the temporary table.

Hive considers a table to be a collection of files in a directory that bears the table name. 
The CREATE TABLE command creates this directory either directly in the Hive 
warehouse or in a subdirectory, if a nondefault schema is used. The LOAD DATA 
command moves the data to the correct location.

When PROC APPEND is used to append to the Hive table, the Hadoop interface places 
data in a new HDFS file. The interface then issues either the LOAD DATA pattern or the 
LOAD DATA plus INSERT INTO pattern described earlier.

**Examples**

This example creates and loads the FLIGHTS98 HiveServer2 table from the 
SASFLT.FLT98 SAS data set.

```plaintext
libname sasflt 'SAS-library';
libname hdp_air hadoop user=myusr1 pwd=mypwd1 server='hdpcluster' schema=statsdiv;

proc sql;
create table hdp_air.flights98 
   as select * from sasflt.flt98;
quit;
```

This example creates and loads the ALLFLIGHTS HiveServer2 table in 
SEQUENCEFILE format from the SASFLT.ALLFLIGHTS SAS data set.

```plaintext
data hdp_air.allflights (dbcreate_table_opts='stored as sequencefile');
set sasflt.allflights;
run;
```

In this example, the SASFLT.FLT98 SAS data set is appended to the ALLFLIGHTS 
HiveServer2 table.

```plaintext
proc append base=hdp_air.allflights 
data=sasflt.flt98;
run;
```
Naming Conventions for SAS and Hive

For general information, see Chapter 2, “SAS Names and Support for DBMS Names.”

Through version .12, Hive is not case sensitive, and all names are lowercase. Therefore, the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options are not necessary. However, beginning with Hive .13, the scope of Hive names has expanded, and these options might be useful. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.)

SAS and Hadoop objects include tables, views, columns, and indexes. Here is how SAS handles Hive names:

- A SAS name must be from 1 to 32 characters long. When Hive column names and table names are 32 characters or less, SAS handles them seamlessly. When SAS reads Hive column names that are longer than 32 characters, a generated SAS variable name is truncated to 32 characters. Hive table names should be 32 characters or less because SAS cannot truncate a table reference. If you already have a table name that is greater than 32 characters, create a Hive table view or use the explicit SQL feature of PROC SQL to access the table.

- If truncating would result in identical names, SAS generates a unique name.

- For National Language Support, because of Hive limitations, column names must use only WLATIN1 characters. Table comments should also contain only WLATIN1 characters when the DBCREATE_TABLE_OPTS= data set option creates them using DBCREATE_TABLE_OPTS="COMMENT 'my table comment'". Although this option accepts any NLS character, the NLS portion of the comment is not displayed properly later.

Data Types for Hadoop

Overview

Hive is a data warehouse that supplies metadata about data that is stored in Hadoop files. Hive includes a data dictionary and an accompanying SQL-like interface called HiveQL or Hive SQL. HiveQL implements data definition language (DDL) and data manipulation language (DML) statements similar to many DBMSs. Hive tables are defined with a CREATE TABLE statement, so every column in a table has a name and a data type. This section includes information about Hive data types and data conversion between Hive and SAS.

For more information about Hive, Hadoop, and data types, see these online documents at https://cwiki.apache.org/confluence/display/Hive.

- Hive Getting Started
- Hive Language Manual
- Hive Plug-in Interfaces — User-Defined Functions and SerDes
- Hive Tutorial
**Supported Hive Data Types**

Here are the Hive data types that the Hadoop engine supports.

- **Numeric data:**
  
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>INT</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>INT</td>
<td>TINYINT</td>
</tr>
</tbody>
</table>

*Note:* When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

- **String data:**
  
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>STRING</td>
</tr>
<tr>
<td>CHARn</td>
<td>VARCHARn</td>
</tr>
</tbody>
</table>

- **Date and time data:**
  
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>TIMESTAMP</td>
</tr>
</tbody>
</table>

- **Complex data:**
  
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td>STRUCT</td>
</tr>
<tr>
<td>MAP</td>
<td></td>
</tr>
</tbody>
</table>

*Note:* In the third maintenance release for SAS 9.4, support for the BINARY and DECIMAL data types was added.

SAS issues CREATE TABLE syntax using VARCHAR instead of STRING. CHAR is supported in Hive .13 and higher.

Prior to Hive .12, dates, times, and timestamps are typically stored in Hive STRING columns in ANSI format. For example, the last day of this millennium is stored as the string ‘2999-12-31’. For Hive releases prior to Hive .12, SAS columns that are formatted as DATE, TIME, or DATETIME are stored in Hive STRING columns.

With Hive .12 and the second maintenance release of SAS 9.4, SAS/ACCESS supports the Hive DATE and TIMESTAMP data types:

With Hive .12 and later, SAS/ACCESS outputs Hive DATE and TIMESTAMP data types for SAS date and SAS datetime formatted columns. As a best practice, use the Hive DATE and TIMESTAMP data types and not the STRING data type.

Hive does not yet support a TIME data type. SAS TIME data types are stored in Hive as STRING data types.
SAS Data Types

SAS has two fundamental data types, character and numeric. SAS character variables (columns) are of a fixed length with a maximum of 32,767 characters. SAS numeric variables are signed eight-byte, floating-point numbers. When SAS numerics are used in conjunction with SAS formats, they can represent a number of data types, including DATE, TIME, and DATETIME. For more detailed information about SAS data types, see *SAS Language Reference: Concepts*.

Data Conversion from Hive to SAS

This table shows the default SAS formats that are assigned to SAS variables that are created when SAS/ACCESS reads Hive table columns.

Note: In the third maintenance release for SAS 9.4, support for BINARY and DECIMAL data types was added.

Table 19.3  Hive to SAS: Default SAS Formats for Hive Data Types

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINAR Y</td>
<td>character</td>
<td>$HEX32767</td>
</tr>
<tr>
<td>CHAR</td>
<td></td>
<td>$255.</td>
</tr>
<tr>
<td>STRING</td>
<td></td>
<td>$32767.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td></td>
<td>$65355</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>numeric</td>
<td>1.</td>
</tr>
<tr>
<td>BIGINT</td>
<td></td>
<td>20.</td>
</tr>
<tr>
<td>DATE</td>
<td></td>
<td>DATE9.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>INT</td>
<td></td>
<td>11.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td></td>
<td>6.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td>DATETIME25.6</td>
</tr>
<tr>
<td>TINYINT</td>
<td></td>
<td>4.</td>
</tr>
</tbody>
</table>
Highlighting New Hive .12 Data Types

This chapter presents considerations for sharing data between Hive and SAS. For example, you need to know how to identify the Hive STRING columns that contain DATE, TIME, or TIMESTAMP information. You also need to know how to minimize the length of SAS character columns that are created from Hive STRING columns. Where you use the VARCHAR, DATE, and TIMESTAMP data types that are available in Hive .12 and later, you optimize sharing data between Hive and SAS. For example, the SASFMT table properties that are described later are not needed for Hive columns that are declared VARCHAR, DATE, and TIMESTAMP.

Information in the rest of this chapter that concerns BIGINT and TIME values remains relevant regardless of whether you use Hive .12 and later.

Highlighting Support for Hive .13 Data Types

Note: In the third maintenance release for SAS 9.4, highlighting support for the BINARY and DECIMAL Hive data types was added.

When specifying a SAS format for a DECIMAL data type with specified precision and scale, specify \texttt{w.d} format where \texttt{w} is the width of the variable including the decimal portion, if any, the decimal point, and a potential negative sign. For example, given a column of type \texttt{DECIMAL(10,0)}, the format is \texttt{11}. Given a column of type \texttt{DECIMAL(18,2)}, the format is \texttt{20.2}.

Issues When Converting Data from Hive to SAS

Below are some potential conversion issues.

- Hive STRING columns that contain ANSI date, time, or timestamp values do not automatically convert respectively to SAS DATE, TIME, or DATETIME types.

- STRING: Depending on the length of Hadoop STRING data, the SAS character $32767. format might be unnecessarily large for short STRING columns or can truncate Hadoop STRING columns that contain more than 32767 characters. To specify a limit for an individual STRING column, issue a Hive ALTER TABLE statement limiting the column length. Here is an example: \texttt{ALTER TABLE weblogs SET TBLPROPERTIES ('SASFMT:webdata'='CHAR(1000)')}. To specify a general limit for multiple STRING columns, use the \texttt{DBMAX\_TEXT=} option.

- BIGINT: Converting Hadoop BIGINT to a SAS numeric can result in loss of precision because the internal SAS eight-byte, floating-point format accurately preserves only 15 digits of precision. A BIGINT preserves up to 19.

Work-arounds are based on how you access data.

- explicitly using pass-through SQL. See “Address Issues When Converting Data from Hive to SAS for Pass-Through SQL” on page 624.

- using the LIBNAME statement. See “Address Issues When Converting Data from Hive to SAS with Table Properties” on page 622.
SAS Table Properties for Hive and Hadoop

Although HiveQL supplies critical metadata for Hadoop files, in some cases more metadata is beneficial. Fortunately, HiveQL CREATE TABLE and ALTER TABLE statements provide an extensible feature called table properties. For more information, see the Hive Language Manual at https://cwiki.apache.org/confluence/display/Hive.

SAS/ACCESS uses table properties to describe and interpret the contents of Hive STRING columns.

Here is an example of a new Hive table that is created with SAS/ACCESS. A SAS variable (column) has an associated SAS DATETIME format, so SAS/ACCESS creates a DATETIME table property for the Hive column.

```sas
libname hdp hadoop server=dbihadoop user=myusr1 pwd=mypwd1;
data hdp.datetime_tableproperty_sample;
    format dt_stamp datetime25.6;
    dt_stamp=datetime();
run;
```

This code creates a new Hive table, DATETIME_TABLEPROPERTY_SAMPLE, by generating this HiveQL:

```
CREATE TABLE `DATETIME_TABLEPROPERTY_SAMPLE` (`dt_stamp` STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'
  STORED AS TEXTFILE TBLPROPERTIES ('SASFMT:dt_stamp'='DATETIME(25.6)')
```

SAS stores `dt_stamp` as a Hive ANSI STRING, as in this example:

```
2012-02-23 09:51:37.218
```

Based on the SAS DATETIME25.6 format, SAS/ACCESS also generates the Hive table property that describes STRING column `dt_stamp` as DATETIME(25.6).

When SAS/ACCESS reads this Hive table, the SASFMT table property indicates STRING column `dt_stamp` contains an ANSI timestamp. SAS/ACCESS automatically converts and formats it a SAS DATETIME25.6 variable, as in this example:

```sas
data;
set hdp.datetime_tableproperty_sample;
put dt_stamp=;
run;
```

```
dt_stamp=23FEB2012:09:51:37.218000
NOTE: There were 1 observations read from the data set HDF.DATETIME_TABLEPROPERTY_SAMPLE.
```

When SAS/ACCESS creates a new Hive table, it generates table properties for SAS variables with character, date, datetime, and time formats—all of which produce Hive STRING columns. See the generated table properties in Data Conversion from SAS to Hive on page 619.

Data Conversion from SAS to Hive

This table shows the Hive data types and table properties that are assigned when SAS/ACCESS creates a Hive table.
SAS sets table properties only when creating a new Hive table. It does not create or alter table properties when appending to an existing Hive table.

Table 19.4  Hive Data Types and Table Properties That Are Created for SAS Data Type and Format Combinations

<table>
<thead>
<tr>
<th>SAS Data Type</th>
<th>SAS Format</th>
<th>Hive Data Type</th>
<th>Hive Table Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>$n.$</td>
<td>STRING</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td></td>
<td>$HEX32767</td>
<td>BINARY</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$n.$</td>
<td>STRING</td>
<td>VARCHAR(n)</td>
</tr>
<tr>
<td>numeric</td>
<td>DATETIMEw.p</td>
<td>STRING</td>
<td>DATETIME(w.p)</td>
</tr>
<tr>
<td></td>
<td>DATEw.</td>
<td>STRING</td>
<td>DATE(w:0)</td>
</tr>
<tr>
<td></td>
<td>TIMEw.p.</td>
<td>STRING</td>
<td>TIME(w:p)</td>
</tr>
<tr>
<td></td>
<td>1. to 4.</td>
<td>SMALLINT</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>5. to 9.</td>
<td>INT</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>other numeric formats</td>
<td>DOUBLE</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>w.d</td>
<td>DECIMAL</td>
<td>none</td>
</tr>
</tbody>
</table>

Leverage Table Properties for Existing Hive Tables

SAS/ACCESS generates SAS table properties only when creating a new Hive table. Many or perhaps all of your Hive tables are created by other means. For example, your Hadoop administrator might create Hive table definitions by submitting DDL scripts to the Hive CLI. SAS and SAS users can benefit by adding SAS table properties to existing Hive table definitions. In this example, a Hive table has already been defined.

```sql
CREATE EXTERNAL TABLE weblogs (extract_date STRING, 
    extract_type INT, webdata STRING) 
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' 
    STORED AS TEXTFILE LOCATION '/user/hadoop/web_data'
```

Based on this table definition, here is how SAS interprets the columns.

```sas
libname hdp hadoop server=mysrv1 user=myusr1 pwd=mypwd1;
data sheetmetal_sales; set hdp.weblogs(obs=1);
put extract_date= extract_type=;
put webdata=;
run;

extract_date=2012-02-21 extract_type=1
webdata=http://www.sas.com/industry/oilgas
NOTE: There were 1 observations read from the data set HDP.WEBLOGS.
```

```sas
proc contents data=hdp.weblogs; run;
```
Notice that Hive describes the `extract_date` column to SAS as a 32767 length STRING. It also describes the `webdata` column as a 32767 length STRING. So SAS/ACCESS enters both of these columns as character data and uses $32767. to format them. The result is an overly wide SAS data set with an observation (row) width of 64 kilobytes that also does not format `extract_date` to a SAS DATE.

SAS issues a warning message for this situation, which includes the maximum column length that was in the result set. In the example, the maximum length read for the `extract_date` STRING column is 10 bytes. The maximum length read for the `webdata` STRING column was 320 bytes.

**WARNING:** SAS/ACCESS assigned these columns a length of 32767. If resulting SAS character variables remain this length, SAS performance is impacted. See SAS/ACCESS documentation for details. Columns followed by the maximum length observed were: `extract_date:10, webdata:320`

The example below assumes that the length of the `webdata` STRING in Hive never exceeds 1000 characters. A Hadoop user ID with the appropriate authority can issue Hive ALTER TABLE statements to add SAS table properties to the Hive table definition.

```sql
ALTER TABLE weblogs SET TBLPROPERTIES ('SASFMT:extract_date'='DATE(9.0)')
ALTER TABLE weblogs SET TBLPROPERTIES ('SASFMT:webdata'='CHAR(1000)')
```

SAS/ACCESS honors the added properties, and here is the result.

```sas
libname hdp hadoop server=mysrv1 user=myusr1 pwd=mypwd1;
data sheetmetal_sales; set hdp.weblogs(obs=1);
put extract_date= extract_type=;
put webdata=;
run;
extract_date=21FEB2012 extract_type=1
webdata=http://www.sas.com/industry/oilgas
NOTE: There were 1 observations read from the data set HDP.WEBLOGS.
```

The resulting SAS data set that is created from the Hive table has a much smaller observation width, which helps SAS save disk space and reduce CPU consumption. It also automatically converts and formats `extract_date` to SAS standard DATE9. format.

Adding SAS properties to existing Hive tables does not impact table use by software that is not SAS. You can also issue ALTER TABLE and other DDL statements using SAS/ACCESS explicit SQL. (See “SQL Pass-Through Facility Specifics for Hadoop”
on page 611). Issuing such DDL as an ALTER TABLE statement can be restricted to only the Hadoop administrator.

**Address Issues When Converting Data from Hive to SAS with Table Properties**

Some issues currently exist when reading Hadoop data into SAS. (See “Issues When Converting Data from Hive to SAS” on page 618.) For example: Hive STRING columns default to the $32767. SAS character format without a defined SASFMT table property or a SAS override option such as DBSASTYPE=.

Here is how you can address specific conversion issues.

**STRING issues**

To automatically convert Hive STRING columns that contain ANSI date, timestamp, or time values to suitable SAS formats, you can use the following HiveQL ALTER TABLE statements. In the statements are these sample Hive columns: `d` contains ANSI date, `ts` contains ANSI timestamp, and `t` contains ANSI time.

**Note:** Keep in mind that you cannot run ALTER TABLE commands in SAS because they are not SAS syntax. You must execute these commands within Hive.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:d'='DATE(9.0)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:ts'='DATETIME(25.6)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:t'='TIME(15.6)')
```

Instead, you could use these statements to create SAS character variables of optimal length that contain the identical ANSI representation as those that are stored in Hive:

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:d'='CHAR(9)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:ts'='CHAR(25)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:t'='CHAR(15)')
```

You can use the following statement for other Hive STRING columns where the maximum length is less than 32767. Here, the `string_col` column has a maximum length of 100.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:string_col'='CHAR(100)')
```

However, if you anticipate that the `string_col` column in Hive might grow to a maximum length of 200 in the future, you could instead use this statement to set the table property.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:string_col'='CHAR(200)')
```

Hive STRING columns longer than 32767 characters are truncated when they are read into SAS. Here is how the warning for this data loss is flagged in the SAS log:

```
WARNING: Column 'string_col' was truncated 1 times.
Observation (row) number 2 was the first observation truncated.
```

**BIGINT issues**

Converting a Hadoop BIGINT column to a SAS numeric column can cause a loss of precision. A SAS numeric column can accurately preserve only 15 digits of precision. However, a BIGINT column can preserve up to 19 significant digits of precision, plus one character for the possible sign (+/-). You can address this issue by applying a CHAR(20) table property format. SAS then automatically reads a Hive BIGINT column into a SAS character string with $20. format. This format preserves all BIGINT digits in character format. Here is an example, using the `bgint` BIGINT column.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:bgint'='CHAR(20)')
```
Keep this important consideration in mind, however: For Hive tables that SAS/ACCESS creates, you might not need to issue ALTER TABLE statements. See “Data Conversion from SAS to Hive” on page 619 for table properties that SAS/ACCESS automatically generates when it creates a Hive table.

**CAUTION:**
- Do not create multiple table properties for a single Hive column.
- Unpredictable data conversion can result.

### Alternatives to Table Properties for Issues with Data Conversion from Hive to SAS

For various reasons, it might be impractical or undesirable to issue ALTER TABLE statements to create SAS table properties. In such cases, you can instead use these data set options.

**DBSASTYPE=**

Use DBSASTYPE= in your SAS code to cause data conversion from Hive to SAS that is identical to automatic conversion with table properties. The pairs below are SAS DATA steps with identical behavior. The first of each pair assumes a SASFMT table property, the second one assumes no table property, and DBSASTYPE= is added to achieve the same functionality. (For details, see the DBSASTYPE= data set option.)

Here is the SAS LIBNAME statement for all of these SAS DATA steps.

**Note:** Remember that you cannot run ALTER TABLE commands in SAS because they are not SAS syntax. You must execute these commands within Hive.

```sas
libname hdp hadoop server=mysrv1 user=myusr1 pwd=mypwd1;

ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:d'='DATE(9.0)')
[---assumes table property 'SASFMT:d'='DATE(9.0)' ---]
data work.local_sample; set hdp.sample_table( keep=d ); run;

[---assumes no table property for column 'd'---]
data work.local_sample;
set hdp.sample_table( keep=d dbastype=(d='DATE(9.0)') ); run;

ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:ts'='DATETIME(25.6)')
[---assumes table property 'SASFMT:ts'='DATETIME(25.6)' ---]
data work.local_sample; set hdp.sample_table( keep=ts ); run;

[---assumes no table property for column 'ts'---]
data work.local_sample;
set hdp.sample_table( keep=ts dbastype=(ts='DATETIME(25.6)') ); run;

ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:t'='TIME(15.6)')
[---assumes table property 'SASFMT:t'='TIME(15.6)' ---]
data work.local_sample; set hdp.sample_table( keep=t ); run;

[---assumes no table property for column 't'---]
data work.local_sample;
set hdp.sample_table( keep=t dbastype=(t='TIME(15.6)') ); run;

ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:string_col'='CHAR(200)')
[---assumes table property 'SASFMT:string_col'='CHAR(200)' ---]
```
data work.local_sample; set hdp.sample_table( keep=string_col ); run;

[---assumes no table property for column ‘string_col’---]
data work.local_sample;
set hdp.sample_table( keep=string_col dbsastype=(string_col='CHAR(200)') );
run;

DBMAX_TEXT=[$n]
You can use the DBMAX_TEXT= option to limit the SAS length of all STRING columns read from Hive. For example, if you set DBMAX_TEXT=100, then all SAS character variables that are created from Hive STRING columns are limited to width $100. Setting the DBMAX_TEXT= option likewise limits the length in SAS of Hive 12 and higher CHAR and VARCHAR columns.

COMPRESSION=YES
If you can use neither table properties nor DBSASTYPE= to reduce the default 32767 SAS character length for Hive STRING columns, consider using this data set option. Although SAS character variables still use the $32767 format, using SAS data set compression, you can realize significant savings in terms of disk space and CPU consumption. Here is an example.

data work.local_sample( COMPRESSION=YES ); set hdp.sample_table; run;

Apply COMPRESSION=YES to the SAS data set reference, not to the Hadoop table reference. Consider using this value even if you reduce default SAS character 32767 lengths with table properties or the DBSASTYPE= option. Also, consider using COMPRESSION=YES when many strings in the Hadoop data are much smaller than the reduced SAS character variable length. For example, you could use COMPRESSION=YES if you had the table property shown below, but most values of address_1 are less than 100 characters.

ALTER TABLE sample_table SET TBLPROPERTIES ({'SASFMT:address_1'='CHAR(1000)'}

For more information about this data set option, see SAS Data Set Options: Reference.

Address Issues When Converting Data from Hive to SAS for Pass-Through SQL

Neither table properties nor DBSASTYPE= address data conversion issues from Hive to SAS if you use pass-through SQL to read Hive data. For pass-through SQL, you might need to explicitly convert and format each Hive column as you want it to be represented in SAS. You can see this if you use SAS to create a table with SAS table properties that are generated for all but the BIGINT column. Here is the table that SAS creates.

libname hdp hadoop server=mysrv1 user=myusr1 pwd=mypwd1;
data hdp.passsthrough_ex( dbtype=(bgint="BIGINT") );
bgint='1234567890123456789';
format ts datetime25.6; ts=datetime();
format d date9.; d=today();
format t time10.; t=time();
format string_col $20.; string_col='hello';
run;

SAS issues this HiveQL when creating the table.

CREATE TABLE `PASSTHROUGH_EX` (`bgint` BIGINT,`ts` STRING,
`d` STRING,`t` STRING,`string_col` STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' \
001' STORED AS TEXTFILE TBLPROPERTIES
Next, an ALTER TABLE statement is issued to add a table property for BIGINT column bgint.

```
ALTER TABLE passthrough_ex SET TBLPROPERTIES ('SASFMT:bgint'='CHAR(20)')
```

A LIBNAME-based table that is read to SAS recognizes the table properties.

```
data work.local; set hdp.passthrough_ex; run;

data _null_; set work.local;
  put bgint=; put ts=; put d=; put t=; put string_col=;
run;
  bgint=1234567890123456789
  ts=25FEB2012:02:00:55.141000
  d=25FEB2012
  t=2:00:55
  string_col=hello
```

This pass-through SQL step converts and formats each column identically to the LIBNAME-based step that applied the table properties.

```
proc sql; connect to hadoop(server=mysrv1 user=myusr1 pwd=mypwd1);
create table work.local as select
  bgint length 20 format $20. informat $20.,
  input(ts, IS8601DT26.) as ts format datetime25.6 informat datetime25.6,
  input(d, yymmdd10.) as d format date9. informat date9.,
  input(t, IS8601TM15.) as t format time15.6 informat time15.6,
  string_col length 20 format $20. informat $20.
from connection to hadoop( select cast(bgint as STRING)
  as bgint,ts,d,t,string_col from passthrough_ex );
quit;
data _null_; set work.local;
  put bgint=; put ts=; put d=; put t=; put string_col=;
run;
  bgint=1234567890123456789
  ts=25FEB2012:02:00:55.141000
  d=25FEB2012
  t=2:00:55.141000
  string_col=hello
```

If SAS detects that a column length for a numeric variable is 32767 and could be less, it writes a message:

```
WARNING: These columns could have a length in SAS of 32767. If so, SAS performance is impacted. See SAS/ACCESS documentation for details. The columns read from Hive followed by the maximum length observed were: bgint:20, ts:26, d:9, t:15, string_col:20.
```

**Hadoop Null Values**

Hadoop has a special value called NULL. A Hadoop NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Hadoop NULL value, it interprets it as a SAS missing value. For more information about
how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

---

**Sample Code for Hadoop**

**Code Snippets**

The code snippets in this section resemble those for most other SAS/ACCESS interfaces.

This snippet shows a list of available Hive tables.

```sas
proc datasets lib=hdp; quit;
```

Here is the metadata for the mytab Hive table.

```sas
proc contents data=hdp.mytab; quit;
```

This snippet extracts mytab data into SAS.

```sas
data work.a;
set hdp.mytab;
run;
```

This extracts a subset of the mytab rows and columns into SAS. Subsetting the rows (with a WHERE statement, for example) can help avoid extracting too much data into SAS.

```sas
data work.a;
set hdp.mytab (keep=col1 col2);
where col2=10;
run;
```

**Use DBSASTYPE= to Load Hadoop Data into SAS**

This example uses the **DBSASTYPE=** data set option to load Hadoop textual dates, timestamps, and times into the corresponding SAS DATE, DATETIME, and TIME formats. The first step reads in a SAS character string to display the data and make clear what occurs in successive steps.

```sas
data; set hdp.testHiveDate; put dt; run;
```

```
2011-10-17
2009-07-30 12:58:59
11:30:01
```

```sas
data; set hdp.testHiveDate(dbsastype=(dt='date')); put dt; run;
```

```
17OCT2011
30JUL2009
```

```sas
data; set hdp.testHiveDate(dbsastype=(dt='datetime')); put dt; run;
```
This code uses SAS SQL to access a Hadoop table.

```sas
proc sql;
create table work.a as select * from hdp.newtab;
quit;
```

SAS data is then loaded into Hadoop.

```sas
data hdp.newtab2;
set work.a;
run;
```

Use implicit pass-through SQL to extract only 10 rows from the Newtab table and load the work SAS data set with the results.

```sas
proc sql;
connect to hadoop server=hxpduped user=myusr1 password=mypwd1;
create table work.a as
    select * from connection to hadoop (select * from newtab limit 10);
```

---

**Create a Partitioned Table with a File Type of SEQUENCEFILE**

Use the `DBCREATE_TABLE_OPTS` value `PARTITIONED BY (column data-type)` `STORED AS SEQUENCEFILE`.

```sas
libname hdp HADOOP server=hxpduped user=myusr1 password=mypwd1;

data hdp.part_tab (DBCREATE_TABLE_OPTS="PARTITIONED BY (s2 int, s3 string) STORED AS SEQUENCEFILE");
set work.part_tab;
run;
```

---

**Using SAS In-Database Processing with Hadoop**

Before you can work with SAS High-Performance Analytics procedures using SAS/ACCESS Interface to Hadoop, you must first install the SAS Embedded Process on your Hadoop cluster. The SAS Embedded Process works with MapReduce projects on your HDFS data in parallel to the analytics grid. For installation information, see *SAS In-Database Products: Administrator's Guide*. 

---

```sas
data; set hdp.testHiveDate(dbsastype=(dt='time'));
put dt;
run;
```
Chapter 20
SAS/ACCESS Interface to HAWQ

Introduction to SAS/ACCESS Interface to HAWQ ........................................ 630
LIBNAME Statement Specifics for HAWQ .................................................. 630
  Overview ................................................................. 630
  Arguments .............................................................. 630
  LIBNAME Statement Examples ............................................... 633
Data Set Options for HAWQ ................................................................. 634
SQL Pass-Through Facility Specifics for HAWQ ....................................... 636
  Key Information ......................................................... 636
  CONNECT Statement Example .......................................... 636
  Special Catalog Queries ................................................ 637
Autopartitioning Scheme for HAWQ ...................................................... 638
  Overview ................................................................. 638
  Autopartitioning Restrictions .......................................... 638
  Nullable Columns ....................................................... 638
  Using WHERE Clauses .................................................. 638
  Using DBSLICEPARM= ............................................... 638
  Using DBSLICE= ....................................................... 638
Passing SAS Functions to HAWQ ............................................................. 640
Passing Joins to HAWQ ....................................................................... 640
Bulk Loading for HAWQ ..................................................................... 641
  Overview ................................................................. 641
  Using Protocols to Access External Tables .................................. 641
  Configuring the File Server ................................................ 642
  Stopping gpfdist ................................................................ 642
  Troubleshooting gpfdist .................................................... 643
  Using the file:// Protocol ................................................ 643
  Accessing Dynamic Data in Web Tables ..................................... 643
  Data Set Options for Bulk Loading ......................................... 643
  Examples ........................................................................ 644
Naming Conventions for HAWQ .............................................................. 644
Data Types for HAWQ ..................................................................... 645
  Overview ................................................................. 645
  Supported HAWQ Data Types .......................................... 645
  HAWQ Null Values ........................................................ 645
  LIBNAME Statement Data Conversions ................................... 646
Introduction to SAS/ACCESS Interface to HAWQ

For available SAS/ACCESS features, see HAWQ supported features on page 81. For more information about HAWQ, see your HAWQ documentation.

SAS/ACCESS Interface to HAWQ and SAS/ACCESS Interface to Greenplum both use the underlying Greenplum ODBC client. For this reason, you might see messages that refer to Greenplum in the SAS log.

LIBNAME Statement Specifics for HAWQ

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to HAWQ supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing HAWQ.

```
LIBNAME libref hawq <connection-options> <LIBNAME-options>;
```

Arguments

```
libref
```

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

```
hawq
```

specifies the SAS/ACCESS engine name for the HAWQ interface.

```
connection-options
```

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to the HAWQ database in two ways. Specify only one of these methods for each connection because they are mutually exclusive.

- SERVER=, DATABASE=, PORT=, USER=, PASSWORD=
- DSN=, USER=, PASSWORD=

Here is how these options are defined.

```
SERVER=<>server-name<>:
```

specifies the HAWQ server name or the IP address of the server host. If the server name contains spaces or nonalphanumeric characters or if it is an IP address, you must enclose it in quotation marks.

```
DATABASE=<>database-name<>:
```

specifies the HAWQ database that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: DB=
PORT=port
  specifies the port number that is used to connect to the specified HAWQ
database. If you do not specify a port, the default is 5432.

USER='<HAWQ-user-name>'
  specifies the HAWQ user name (also called the user ID) that is used to connect to
the database. If the user name contains spaces or nonalphanumeric characters,
use quotation marks.

PASSWORD='<HAWQ-password>'
  specifies the password that is associated with your HAWQ user ID. If the
password contains spaces or nonalphabetic characters, you must enclose it in
quotation marks. You can also specify PASSWORD= with the PWD=, PASS=,
and PW= aliases.

DSN='<HAWQ-data-source>'
  specifies the configured HAWQ ODBC data source to which you want to
connect. It is recommended that you use this option only if you have configured
HAWQ ODBC data sources on your client. This method requires additional setup
—either through the ODBC Administrator control panel on Windows platforms,
or through the odbc.ini file or a similarly named configuration file on UNIX
platforms. It is recommended that you use this connection method only if you
have existing, functioning data sources that have been defined.

LIBNAME-options
  define how SAS processes DBMS objects. Some LIBNAME options can enhance
performance, and others determine locking or naming behavior. The following table
describes the LIBNAME options for SAS/ACCESS Interface to HAWQ with the
applicable default values. For details, see LIBNAME Options for Relational
Databases on page 101.

Table 20.1  SAS/ACCESS LIBNAME Options for HAWQ

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>operation-specific</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>FORWARD_ONLY</td>
</tr>
</tbody>
</table>
| DBCOMMIT=                | 1000 (when inserting rows), 0 (when
                              updating rows)                   |
<p>| DBCONINIT=               | none                               |
| DBCONTERM=               | none                               |
| DBCREATE_TABLE_OPTS=     | none                               |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DLBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DLBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>none</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>none</td>
</tr>
<tr>
<td>DBASASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS</td>
</tr>
<tr>
<td>DEFER=</td>
<td>none</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>none</td>
</tr>
<tr>
<td>IN=</td>
<td>none</td>
</tr>
<tr>
<td>INSERT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>KEYSET_SIZE=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for HAWQ”</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>see “Naming Conventions for HAWQ”</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>RC</td>
</tr>
</tbody>
</table>
LIBNAME Statement Specifics for HAWQ

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>READBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SQLGENERATION=</td>
<td>DBMS</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>none</td>
</tr>
<tr>
<td>TRACE=</td>
<td>none</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>RC</td>
</tr>
<tr>
<td>UPDATE_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>none</td>
</tr>
</tbody>
</table>

**LIBNAME Statement Examples**

In this example, SERVER=, DATABASE=, PORT=, USER=, and PASSWORD= are the connection options.

```plaintext
libname mydblib hawq server=hwq04 db=customers port=5432
  user=hwqusr1 password=hwqpwd1;

proc print data=mydblib.customers;
  where state='CA';
run;
```

In the next example, DSN=, USER=, and PASSWORD= are the connection options. The HAWQ data source is configured in the ODBC Administrator Control Panel on Windows platforms. It is also configured in the odbc.ini file or a similarly named configuration file on UNIX platforms.

```plaintext
libname mydblib hawq DSN=hwqSalesDiv user=hwqusr1 password=hwqpwd1;

proc print data=mydblib.customers;
  where state='CA';
run;
```
Data Set Options for HAWQ

All SAS/ACCESS data set options in this table are supported for HAWQ. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

Table 20.2 SAS/ACCESS Data Set Options for HAWQ

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_DATAFILE_EXISTS=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DELTE_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DELIMITER=</td>
<td></td>
</tr>
<tr>
<td>BL_ENCODING=</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>BL_ESCAPE=</td>
<td>\</td>
</tr>
<tr>
<td>BL_EXCEPTION=</td>
<td>none</td>
</tr>
<tr>
<td>BL_EXECUTE_CMD=</td>
<td>none</td>
</tr>
<tr>
<td>BL_EXECUTE_LOCATION=</td>
<td>none</td>
</tr>
<tr>
<td>BL_EXTERNAL_WEB=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_FORCE_NOT_NULL=</td>
<td>none</td>
</tr>
<tr>
<td>BL_FORMAT=</td>
<td>TEXT</td>
</tr>
<tr>
<td>BL_HEADER=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_HOST=</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>BL_NULL=</td>
<td>'\N' [TEXT mode], unquoted empty value [CSV mode]</td>
</tr>
<tr>
<td>BL_PORT=</td>
<td>8080</td>
</tr>
<tr>
<td>BL_PROTOCOL=</td>
<td>'gpfdist'</td>
</tr>
<tr>
<td>BL_QUOTE=</td>
<td>&quot; (double quotation mark)</td>
</tr>
<tr>
<td>BL_REJECT_LIMIT=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>BL_REJECT_TYPE</td>
<td>ROWS</td>
</tr>
<tr>
<td>BULKLOAD</td>
<td>none</td>
</tr>
<tr>
<td>DBCOMMIT</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL</td>
<td>none</td>
</tr>
<tr>
<td>DBMASTER</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL</td>
<td>none</td>
</tr>
<tr>
<td>DBNULLKEYS</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASTYPE</td>
<td>see “Data Types for Greenplum”</td>
</tr>
<tr>
<td>DBSLICE</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM</td>
<td>THREADED_APPS</td>
</tr>
<tr>
<td>DBTYPE</td>
<td>see “Data Types for Greenplum”</td>
</tr>
<tr>
<td>DISTRIBUTED_BY</td>
<td>DISTRIBUTEDRANDOMLY</td>
</tr>
<tr>
<td>ERRLIMIT</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS</td>
<td>none</td>
</tr>
<tr>
<td>INSERTBUFF</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL</td>
<td>a blank character</td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for HAWQ

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the HAWQ interface.

- The `dbms-name` is **HAWQ**.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to HAWQ. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default **HAWQ** alias is used.
- The CONNECT statement `database-connection-arguments` are identical to its LIBNAME connection options.

**CONNECT Statement Example**

This example uses the DBCON alias to connect to the **HAWQ04** HAWQ server database and execute a query. The connection alias is optional.

```sas
proc sql;
   connect to hawq as dbcon
   (server=HAWQ04 db=sample port=5432 user=hwqusr1 password=hwqpwd1);
   select * from connection to dbcon
   (select * from customers where customer like '1%');
quit;
```
**Special Catalog Queries**

SAS/ACCESS Interface to HAWQ supports the following special queries. You can use the queries to call functions in ODBC-style function application programming interfaces (APIs). Here is the general format of the special queries:

\[
\text{HAWQ::SQLAPI } \text{'parameter-1', 'parameter-n'}
\]

HAWQ:: is required to distinguish special queries from regular queries. HAWQ:: is not case sensitive.

SQLAPI is the specific API that is being called. SQLAPI is not case sensitive.

\text{'parameter n'}

is a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters, and the underscore represents any single character. To use either character as a literal value, you can use the backslash character (\) to escape the match characters. For example, this call to SQLTables usually matches table names such as myatest and my_test:

\[
\text{select * from connection to hawq}
\]

\[
\text{(HAWQ::SQLTables "test","","my_test");}
\]

Use the escape character to search only for the my_test table:

\[
\text{select * from connection to hawq}
\]

\[
\text{(HAWQ::SQLTables "test","","my\_test");}
\]

SAS/ACCESS Interface to HAWQ supports these special queries.

\[
\text{HAWQ::SQLTables } \text{<'Catalog', 'Schema', 'Table-name', 'Type'>}
\]

returns a list of all tables that match the specified arguments. If you do not specify any arguments, all accessible table names and information are returned.

\[
\text{HAWQ::SQLColumns } \text{<'Catalog', 'Schema', 'Table-name', 'Column-name'>}
\]

returns a list of all columns that match the specified arguments. If you do not specify any arguments, all accessible column names and information are returned.

\[
\text{HAWQ::SQLColumns } \text{<'Catalog', 'Schema', 'Table-name', 'Column-name'>}
\]

returns a list of all columns that match the specified arguments. If you do not specify any argument, all accessible column names and information are returned.

\[
\text{HAWQ::SQLPrimaryKeys } \text{<'Catalog', 'Schema', 'Table-name"Type'>}
\]

returns a list of all columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If you do not specify any table name, this special query fails.

\[
\text{HAWQ::SQLStatistics } \text{<'Catalog', 'Schema', 'Table-name'>}
\]

returns a list of the statistics for the specified table name. You can set the SQL_INDEX_ALL and SQL_ENSURE options in the SQLStatistics API call. If you do not specify any table name argument, this special query fails.

\[
\text{HAWQ::SQLGetTypeInfo}
\]

returns information about the data types that the HAWQ database supports.
Autopartitioning Scheme for HAWQ

Overview

Autopartitioning for SAS/ACCESS Interface to HAWQ is a modulo (MOD) function method. For general information about this feature, see "Autopartitioning Techniques in SAS/ACCESS" on page 59.

Autopartitioning Restrictions

SAS/ACCESS Interface to HAWQ places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here is how columns are partitioned.

• INTEGER and SMALLINT columns are given preference.
• You can use other numeric columns for partitioning if the precision minus the scale of the column is greater than 0 but less than 10—namely, 0<\(\text{precision-scale}\)<10.

Nullable Columns

If you select a nullable column for autopartitioning, the OR\(<\text{column-name}\>\text{IS NULL}\) SQL statement is appended at the end of the SQL code that is generated for the threaded Read. This ensures that any possible NULL values are returned in the result set.

Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in a SAS WHERE clause. For example, this DATA step cannot use a threaded Read to retrieve the data because all numeric columns in the table are in the WHERE clause:

```
data work.locemp;
set trlib.MYEMPS;
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

Using DBSLICEPARM=

Although SAS/ACCESS Interface to HAWQ defaults to three threads when you use autopartitioning, do not specify a maximum number of threads for the threaded Read in DBSLICEPARM= LIBNAME option on page 151.

Using DBSLICE=

You might achieve the best possible performance when using threaded Reads by specifying the DBSLICE= data set option for HAWQ in your SAS operation. This is especially true if your HAWQ data is evenly distributed across multiple partitions in a HAWQ database system.
When you create a HAWQ table using the HAWQ database partition model, you can specify the partitioning key that you want to use by appending the `PARTITION BY <column-name>` clause to your CREATE TABLE statement. Here is how you can accomplish this by using the `DB_CREATE_TABLE_OPTS=LIBNAME` option within the SAS environment.

```sas
/* Points to a triple-node server. */
libname mylib hawq server=hawq03 user=myuser pw=mypwd db=HAWQ;
DBCREATE_TABLE_OPTS='PARTITION BY (EMPNUM);

proc datasets library=mylib;
delete MYEMPS1;run;
data mylib.myemps(drop=morf whatstate
DBTYPE=(HIREDATE="date" SALARY="numeric(8,2)"
NUMCLASS="smallint" GENDER="char(1)" ISTENURE="numeric(1)" STATE="char(2)"
EMPNUM="int NOT NULL Primary Key");
format HIREDATE mmddyy10.;
do EMPNUM=1 to 100;
morf=mod(EMPNUM,2)+1;
if (morf eq 1) then
   GENDER='F';
else
   GENDER='M';
SALARY=(ranuni(0)*5000);
HIREDATE=int(ranuni(13131)*3650);
whatstate=int(EMPNUM/5);
if (whatstate eq 1) then
   STATE='FL';
if (whatstate eq 2) then
   STATE='GA';
if (whatstate eq 3) then
   STATE='SC';
if (whatstate eq 4) then
   STATE='VA';
else
   state='NC';
ISTENURE=mod(EMPNUM,2);
NUMCLASS=int(EMPNUM/5)+2;
output;
end;
run;
```

After the MYEMPS table is created on this three-node database, a third of the rows reside on each of the three nodes.

Using `DBSLICE=` works well when the table that you want to read is not stored in multiple partitions. It gives you flexibility in column selection. For example, if you know that the `STATE` column in your employee table contains only a few distinct values, you can modify your `DBSLICE=` option accordingly.

```sas
data work.locemp;
set mylib.MYEMPS (DBSLICE= ("STATE='GA'"
   "STATE='SC'" "STATE='VA'" "STATE='NC'");
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```
Passing SAS Functions to HAWQ

SAS/ACCESS Interface to HAWQ passes the following SAS functions to HAWQ for processing. Where the HAWQ function name differs from the SAS function name, the HAWQ name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>HAWQ Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>MAX</td>
</tr>
<tr>
<td>ARCOS (ACOS)</td>
<td>MIN</td>
</tr>
<tr>
<td>ARSIN (ASIN)</td>
<td>MINUTE (DATEPART)</td>
</tr>
<tr>
<td>ATAN</td>
<td>MONTH (DATEPART)</td>
</tr>
<tr>
<td>ATAN2</td>
<td>QTR (DATEPART)</td>
</tr>
<tr>
<td>AVG</td>
<td>REPEAT</td>
</tr>
<tr>
<td>BYTE (CHR)</td>
<td>SECOND (DATEPART)</td>
</tr>
<tr>
<td>CEIL</td>
<td>SIGN</td>
</tr>
<tr>
<td>COMPRESS (TRANSATE)</td>
<td>SIN</td>
</tr>
<tr>
<td>COS</td>
<td>SQRT</td>
</tr>
<tr>
<td>COUNT</td>
<td>STRIP (BTRIM)</td>
</tr>
<tr>
<td>DAY (DATEPART)</td>
<td>SUBSTR (SUBSTRING)</td>
</tr>
<tr>
<td>EXP</td>
<td>SUM</td>
</tr>
<tr>
<td>FLOOR</td>
<td>TAN</td>
</tr>
<tr>
<td>HOUR (DATEPART)</td>
<td>TRANWRD (REPLACE)</td>
</tr>
<tr>
<td>INDEX (STRPOS)</td>
<td>TRIMN (RTRIM)</td>
</tr>
<tr>
<td>LENGTH</td>
<td>UPCASE (UPPER)</td>
</tr>
<tr>
<td>LOG (LN)</td>
<td>WEEKDAY (DATEPART)</td>
</tr>
<tr>
<td>LOG10 (LOG)</td>
<td>YEAR (DATEPART)</td>
</tr>
<tr>
<td>LOWCASE (LOWER)</td>
<td></td>
</tr>
</tbody>
</table>

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to HAWQ. Due to incompatibility in date and time functions between HAWQ and SAS, HAWQ might not process them correctly. Check your results to determine whether these functions are working as expected.

<table>
<thead>
<tr>
<th>Date/Time Function</th>
<th>HAWQ Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE (NOW)</td>
<td>TIME (current_time)</td>
</tr>
<tr>
<td>DATEPART (CONVERT)</td>
<td>TIMEPART (TIME)</td>
</tr>
<tr>
<td>DATETIME (NOW)</td>
<td>TODAY (NOW)</td>
</tr>
</tbody>
</table>

Passing Joins to HAWQ

For a multiple libref join to pass to HAWQ, all of these components of the LIBNAME statements must match exactly.
Bulk Loading for HAWQ

Overview

Bulk loading provides high-performance access to external data sources. Multiple HAWQ instances read data in parallel, which enhances performance.

Bulk loading is the fastest way to insert large numbers of rows into HAWQ tables. You can also use bulk loading to execute high-performance SQL queries against external data sources, without first loading those data sources into a HAWQ database. These fast SQL queries let you optimize extraction, transformation, and loading tasks that are common in data warehousing.

Two types of external data sources, external tables and web tables, have different access methods. External tables contain static data that can be scanned multiple times. The data does not change during queries. Web tables provide access to dynamic data sources as if those sources were regular database tables. Web tables cannot be scanned multiple times. The data can change during the course of a query.

You must specify BULKLOAD=YES to use the bulk-load facility.

The following sections show you how to access external tables and web tables using the bulk-load facility.

Using Protocols to Access External Tables

Use these protocols to access (static) external tables.

gpfdist://

To use the gpfdist:// protocol, install and configure the gpfdist (HAWQ file distribution) program on the host that stores the external tables, see “Configuring the File Server”. The gpfdist utility serves external tables in parallel to the primary HAWQ database segments. The gpfdist:// protocol is advantageous because it ensures that all HAWQ database segments are used during the loading of external tables.

To specify files to gpfdist, use the BL_DATAFILE= data set option. Specify file paths that are relative to the directory from which gpfdist is serving files (the directory where you executed gpfdist).
The gpfdist utility is part of the loader package for the platform where SAS is running. You can also download it from the HAWQ website: http://pivotal.io/.

file://

To use the file:// protocol, external tables must reside on a segment host in a location that HAWQ superusers can access. The segment host name must match the host name, as specified in the gp_configuration system catalog table. In other words, the external tables that you want to load must reside on a host that is part of the set of servers that comprise the database configuration. The file:// protocol is advantageous because it does not require configuration.

**Configuring the File Server**

Follow these steps to configure the gpfdist file server.

2. Define and load a new environment variable called GPLOAD_HOME.
3. Set the value of the variable to the directory that contains the external tables that you want to load.
   The directory path must be relative to the directory in which you execute gpfdist, and it must exist before gpfdist tries to access it.
   - For Windows, open My Computer, select the Advanced tab, and click the Environment Variables button.
   - For UNIX, enter this command or add it to your profile:
     ```
     export GPLOAD_HOME=directory
     ```
4. Start gpfdist as shown in these examples.
   - For Windows:
     ```
     C:> gpfdist -d %GPLOAD_HOME% -p 8080 -l %GPLOAD_HOME%\gpfdist.log
     ```
   - For UNIX:
     ```
     $ gpfdist -d $GPLOAD_HOME -p 8080 -l $GPLOAD_HOME/gpfdist.log &
     ```

You can run multiple instances of gpfdist on the same host as long each instance has a unique port and directory.

If you do not set GPLOAD_HOME, the value of the BL_DATAFILE= data set option specifies the directory that contains the external tables to be loaded. If BL_DATAFILE is not specified, the current directory is assumed to contain the external tables.

**Stopping gpfdist**

In Windows, to stop an instance of gpfdist, use the Task Manager or close the Command Window that you used to start that instance of gpfdist.

Follow these steps in UNIX to stop an instance of gpfdist.

1. Find the process ID:
   ```
   $ ps ax | grep gpfdist (Linux)
   $ ps -ef | grep gpfdist (Solaris)
   ```
2. Kill the process. Here is an example:
   ```
   $ kill 3456
   ```
Troubleshooting gpfdist

Run this command to test connectivity between an instance of gpfdist and a HAWQ database segment.

$ wget http://gpfdist_hostname:port/filename

Using the file:// Protocol

You can use the file:// protocol to identify external files for bulk loading with no additional configuration required. However, using the GPLOAD_HOME environment variable is highly recommended. If you do not specify GPLOAD_HOME, the BL_DATAFILE data set option specifies the source directory. The default source directory is the current directory if you do not set BL_DATAFILE=. The HAWQ server must have access to the source directory.

Accessing Dynamic Data in Web Tables

Use these data set options to access web tables:

- BL_EXECUTE_CMD=
- BL_LOCATION=

Data Set Options for Bulk Loading

Here are the HAWQ bulk-load data set options. For detailed information about these options, see Data Set Options for Relational Databases on page 241.

- BL_DATAFILE=
- BL_CLIENT_DATAFILE=
- BL_DELETE_DATAFILE=
- BL_DELIMITER=
- BL_ENCODING=
- BL_ESCAPE=
- BL_EXCEPTION=
- BL_EXECUTE_CMD=
- BL_EXECUTE_LOCATION=
- BL_EXTERNAL_WEB=
- BL_FORCE_NOT_NULL=
- BL_FORMAT=
- BL_HEADER=
- BL_HOST=
- BL_NULL=
- BL_PORT=
- BL_PROTOCOL=
Examples

This first example shows how you can use a SAS data set, SASFLT.FLT98, to create and load a HAWQ table, FLIGHTS98.

libname sasflt 'SAS-data-library';
libname mydblib hawq server=mysrv1_users
db=users user=myusr1 password=mypwd1;

proc sql;
create table net_air.flights98
   (BULKLOAD=YES
    BL_DATAFILE='c:\temp\HAWQ\data.dat'
    BL_USE_PIPE=NO
    BL_DELETE_DATAFILE=yes
    BL_HOST='192.168.x.x'
    BL_PORT=8081)
   as select * from sasflt.flt98;
quit;

This next example shows how you can append the SAS data set, SASFLT.FLT98, to the existing HAWQ table ALLFLIGHTS. The BL_USE_PIPE=NO option forces SAS/ACCESS Interface to HAWQ to write data to a flat file, as specified in the BL_DATAFILE= option. Rather than deleting the data file, BL_DELETE_DATAFILE=NO causes the engine to leave it after the load has completed.

proc append base=new_air.flights98
   (BULKLOAD=YES
    BL_DATAFILE='c:\temp\HAWQ\data.dat'
    BL_USE_PIPE=NO
    BL_DELETE_DATAFILE=no
    BL_HOST='192.168.x.x'
    BL_PORT=8081)
data=sasflt.flt98;
run;

Naming Conventions for HAWQ

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The HAWQ interface supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name results in identical names, SAS generates a unique name by replacing the last
character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

The `PRESERVE_TAB_NAMES=` and `PRESERVE_COL_NAMES=` options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) HAWQ is not case sensitive, so all names default to lowercase.

HAWQ objects include tables, views, and columns. They follow these naming conventions.

- A name can contain up to 128 characters.
- The first character in a name can be a letter, @, _, or #.
- A name cannot be a HAWQ reserved word, such as WHERE or VIEW.
- A name must be unique within each type of each object.

---

**Data Types for HAWQ**

**Overview**

Every column in a table has a name and a data type. The data type tells HAWQ how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about HAWQ data types, null and default values, and data conversions.

SAS/ACCESS Interface to HAWQ does not directly support any data types that are not listed below. Any columns using these types are read into SAS as character strings.

For more information about HAWQ data types and to determine which data types are available for your version of HAWQ, see your HAWQ documentation.

**Supported HAWQ Data Types**

Here are the data types that SAS/ACCESS Interface to HAWQ supports:

- Character data:
  
  - CHAR(n)
  - VARCHAR(n)
  - TEXT

- Numeric data:
  
  - BIGINT
  - REAL
  - SMALLINT
  - FLOAT
  - INTEGER
  - DECIMAL | DEC | NUMERIC
  - DOUBLE PRECISION

*Note:* When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results,
such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

- Date, time, and timestamp data:
  
  DATE
  TIME
  TIMESTAMP

  Note: Be aware that columns of these data types can contain data values that are out of range for SAS.

- Binary data: BYTEA

**HAWQ Null Values**

HAWQ has a special value called NULL. A HAWQ NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a HAWQ NULL value, it interprets it as a SAS missing value. When loading SAS tables from HAWQ sources, SAS/ACCESS stores HAWQ NULL values as SAS missing values.

In HAWQ tables, NULL values are valid in all columns by default. There are two methods to define a column in a HAWQ table so that it requires data:

- Using SQL, you specify a column as NOT NULL. This tells SQL to allow only a row to be added to a table if a value exists for the field. Rows that contain NULL values in that column are not added to the table.

- Another approach is to assert NOT NULL DEFAULT.

When creating HAWQ tables with SAS/ACCESS, you can use the DBNULL= data set option to specify the treatment of NULL values. For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

Once you know whether a HAWQ column enables NULLs or the host system supplies a default value for a column that is defined as NOT NULL WITH DEFAULT, you can write selection criteria and enter values to update a table. Unless a column is defined as NOT NULL or NOT NULL DEFAULT, it enables NULL values.

To control how the DBMS handles SAS missing character values, use the NULLCHAR= and NULLCHARVAL= data set options.

**LIBNAME Statement Data Conversions**

The following table shows the default formats that SAS/ACCESS Interface to HAWQ assigns to SAS variables when using the LIBNAME statement to read from a HAWQ table.

These default formats are based on HAWQ column attributes.
Table 20.3  LIBNAME Statement: Default SAS Formats for HAWQ Data Types

<table>
<thead>
<tr>
<th>HAWQ Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)*</td>
<td>character</td>
<td>$w$.</td>
</tr>
<tr>
<td>VARCHAR(n)*</td>
<td>character</td>
<td>$w$.</td>
</tr>
<tr>
<td>BYTEA</td>
<td>character</td>
<td>$w^{***}$</td>
</tr>
<tr>
<td>INTEGER</td>
<td>numeric</td>
<td>11.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.</td>
</tr>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>DECIMAL(p,s)**</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>NUMERIC(p,s)**</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME8.</td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME25.6</td>
</tr>
</tbody>
</table>

*  
  * n in HAWQ data types is equivalent to w in SAS formats.
  
  **  
  ** p and s in HAWQ numeric data types are equivalent to w and d in SAS formats.
  
  ***  
  *** Because the Greenplum ODBC driver does the conversion, this field is displayed as if the $HEXw.$ format were applied.

The next table shows the default HAWQ data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

Table 20.4  LIBNAME Statement: Default HAWQ Data Types for SAS Variable Formats

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>HAWQ Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.d</td>
<td>DECIMAL(p,s)**</td>
</tr>
<tr>
<td>$w.$</td>
<td>VARCHAR(n)*</td>
</tr>
<tr>
<td>datetime formats**</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>TIME</td>
</tr>
</tbody>
</table>

*  
  * n in HAWQ data types is equivalent to w in SAS formats.
  
  **  
  ** p and s in HAWQ numeric data types are equivalent to w and d in SAS formats.
Chapter 21
SAS/ACCESS Interface to Impala

Introduction to SAS/ACCESS Interface to Impala ........................................... 650
  Overview ........................................................................................................... 650
  Impala Concepts ............................................................................................. 650
LIBNAME Statement Specifics for Impala .......................................................... 650
  Overview ........................................................................................................... 650
  Arguments ........................................................................................................ 650
  Impala LIBNAME Statement Examples ......................................................... 653
Data Set Options for Impala .............................................................................. 653
SQL Pass-Through Facility Specifics for Impala ............................................... 654
  Key Information .............................................................................................. 654
  CONNECT Statement Examples ...................................................................... 655
Passing SAS Functions to Impala ...................................................................... 655
  Passing Joins to Impala .................................................................................... 656
Bulk Loading for Impala .................................................................................... 656
  Loading ............................................................................................................ 656
  Data Set Options for Bulk Loading ................................................................. 657
  Example 1: Create an Impala Table from a SAS Data Set ............................. 657
  Example 2: Append a SAS Data Set to an Existing Impala Table ............... 657
Naming Conventions for Impala ....................................................................... 658
Data Types for Impala ....................................................................................... 659
  Overview ........................................................................................................... 659
  Supported Impala Data Types ...................................................................... 659
  SAS Data Types .............................................................................................. 659
  Data Conversion from Impala to SAS ............................................................. 660
  Issues When Converting from Impala to SAS ............................................... 660
  Data Conversion from SAS to Impala ............................................................. 661
  Impala Null Values ......................................................................................... 661
Introduction to SAS/ACCESS Interface to Impala

Overview

For available SAS/ACCESS features, see Impala supported features on page 82. For more information about Impala, see your Impala documentation.

Note: The SAS/ACCESS Interface to Impala was implemented in the second maintenance release for SAS 9.4.

Impala Concepts

Cloudera Impala is an open-source, massively parallel processing (MPP) query engine that runs natively on Apache Hadoop. You can use it to issue SQL queries to data stored in HDFS and Apache Hbase without moving or transforming data. Similar to other SAS/ACCESS engines, SAS/ACCESS Interface to Impala lets you run SAS procedures against data that is stored in Impala and returns the results to SAS. You can use it to read and write data to and from Hadoop as if it were any other relational data source to which SAS can connect.

LIBNAME Statement Specifics for Impala

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Impala supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Impala.

LIBNAME libref impala <connection-options> <LIBNAME-options>;

Arguments

libref

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

impala

specifies the SAS/ACCESS engine name for the Impala interface.

connection-options

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. Here is how these options are defined.

Note: All of the following connection options are also valid in the CONNECT statement when you use the SQL pass-through facility (SQL procedure) to connect to your DBMS.

USER='Impala-user-name'

specifies the user name.
PASSWORD='Impala-password'
  specifies the Impala password that is associated with your user ID. If it contains
  spaces or nonalphanumeric characters, you must enclose it in quotation marks. If
  you do not want to enter your Impala password in uncoded text on this statement,
  see PROC PWENCODE in the Base SAS Procedures Guide for a method to
  encode it.

  Alias: PASS=, PWD=, PW=

SERVER='Impala-server-name'
  specifies the Impala server name that runs the Impala daemon. If the server name
  contains spaces or nonalphanumeric characters, you must enclose it in quotation
  marks.

  Alias: HOST=

PORT=port
  specifies the port number that is used to connect to the specified Impala server.

  Default: 21050

SCHEMA=Impala-schema
  specifies the Hive schema.

  Alias: DATABASE=, DB=

  Default: none

CONOPTS='Impala ODBC-connections-options'
  specifies connection options for your data source or database. Separate multiple
  options with a semicolon. Refer to your data source or database’s ODBC driver
  documentation for a list of the ODBC connection options that your ODBC driver
  supports.

LIBNAME-options
  define how SAS processes DBMS objects. The following table describes the
  LIBNAME options for SAS/ACCESS Interface to Impala, with the applicable
default values. This table also identifies LIBNAME options that are valid in the
  CONNECT statement in the SQL procedure. For more information, see LIBNAME
  Options for Relational Databases on page 101 or Chapter 14, “SQL Pass-Through
  Facility for Relational Databases,” on page 471.

Table 21.1 SAS/ACCESS LIBNAME Options for Impala

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>BL_HOST=</td>
<td>SERVER= value, if SERVER= is set; otherwise, none</td>
</tr>
<tr>
<td>BL_PORT=</td>
<td>50070 (required only if Impala HDFS streaming is running on a nondefault port)</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONFIG=</td>
<td>none (SAS_HADOOP_CONFIG_PATH is the default when you set only BULKLOAD=YES)</td>
</tr>
<tr>
<td>CONFIGDIR=</td>
<td>none (SAS_HADOOP_JAR_PATH is the default when you set only BULKLOAD=YES)</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>DRIVER_VENDOR=</td>
<td>CLOUDERA</td>
</tr>
<tr>
<td>HDFS_PRINCIPAL=</td>
<td>none</td>
</tr>
<tr>
<td>IMPALA_PRINCIPAL</td>
<td>= none</td>
</tr>
<tr>
<td>LOGIN_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
</tbody>
</table>
**Impala LIBNAME Statement Examples**

This example uses the default Impala port.

```
libname imp impala server=hxpduped schema=myschema
   user=myusr1 password=mypwd1;
```

This example explicitly specifies the default Impala port.

```
libname imp impala server=hxpduped port=21050 schema=myschema
   user=myusr1 password=mypwd1;
```

---

### Data Set Options for Impala

All SAS/ACCESS data set options in this table are supported for Impala. Default values are provided where applicable. For details, see [Data Set Options for Relational Databases](#) on page 241.

**Table 21.2 SAS/ACCESS Data Set Options for Impala**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_DATAFILE=</td>
<td>automatically generated</td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_HOST=</td>
<td>SERVER= value; if SERVER= is set; otherwise, none</td>
</tr>
<tr>
<td>BL_PORT=</td>
<td>50070</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>CONFIG=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>CONFIGDIR=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_EXTERNAL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBCREATE_TABLE_LOCATION=</td>
<td>/user/hive/warehouse/tabname [with the default schema], /user/hive/warehouse/schema.db/tabname [with a nondefault schema]</td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DBCREATE_TABLE_OPTS=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBGEN_NAME=</code></td>
<td>DBMS</td>
</tr>
<tr>
<td><code>DBLABEL=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBMASTER=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBMAX_TEXT=</code></td>
<td>1024</td>
</tr>
<tr>
<td><code>DBSASLABEL=</code></td>
<td>COMPAT</td>
</tr>
<tr>
<td><code>DBSASTYPE=</code></td>
<td>see Data Types for Impala</td>
</tr>
<tr>
<td><code>DBTYPE=</code></td>
<td>see Data Types for Impala</td>
</tr>
<tr>
<td><code>HDFS_PRINCIPAL=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>PARTITIONED_BY=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>POST_STMT_OPTS=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>POST_TABLE_OPTS=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>PRE_STMT_OPTS=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>PRE_TABLE_OPTS=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>QUERY_TIMEOUT=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>READBUFF=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>SCHEMA=</code></td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>

---

### SQL Pass-Through Facility Specifics for Impala

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Impala interface.

- The `dbms-name` is **IMPALA**.
- The CONNECT statement is required.
• PROC SQL supports multiple connections to Impala. If you use multiple simultaneous connections, you must use the alias argument to identify the different connections. If you do not specify an alias, the default **IMPALA** alias is used.

• The CONNECT statement *database-connection-arguments* are identical to its LIBNAME connection options. Additional LIBNAME options that are valid in the CONNECT statement are indicated in Table 21.1 on page 651.

**CONNECT Statement Examples**

This example uses the default Impala port.

```sql
proc sql;
   connect to impala (user="myusr1" pw="mypwd1" server=hxpduped) schema=myschema;
```

This example explicitly specifies the default Impala port.

```sql
proc sql;
   connect to impala (user="myusr1" pw="mypwd1" server=hxpduped port=21050 schema=myschema schema=default);
```

---

### Passing SAS Functions to Impala

SAS/ACCESS Interface to Impala passes the following SAS functions to Impala for processing. Where the Impala function name differs from the SAS function name, the Impala name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>Impala Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>LOWCASE (LOWER)</td>
</tr>
<tr>
<td>ARCOS (ACOS)</td>
<td>MAX</td>
</tr>
<tr>
<td>ARSIN (ASIN)</td>
<td>MIN</td>
</tr>
<tr>
<td>ATAN</td>
<td>MINUTE</td>
</tr>
<tr>
<td>AVG</td>
<td>MONTH</td>
</tr>
<tr>
<td>CEIL</td>
<td>SECOND</td>
</tr>
<tr>
<td>COS</td>
<td>SIN</td>
</tr>
<tr>
<td>COUNT</td>
<td>SQRT</td>
</tr>
<tr>
<td>DAY</td>
<td>STRIP (TRIM)</td>
</tr>
<tr>
<td>EXP</td>
<td>SUBSTR</td>
</tr>
<tr>
<td>FLOOR</td>
<td>SUM</td>
</tr>
<tr>
<td>HOUR</td>
<td>TAN</td>
</tr>
<tr>
<td>INDEX (LOCATE)</td>
<td>TRANWRD (REGEXP_REPLACE)</td>
</tr>
<tr>
<td>LENGTH</td>
<td>TRIMN (RTRIM)</td>
</tr>
<tr>
<td>LOG (LN)</td>
<td>UPCASE (UPPER)</td>
</tr>
<tr>
<td>LOG10</td>
<td>YEAR</td>
</tr>
</tbody>
</table>

**SQL_FUNCTIONS=ALL** allows for SAS functions that have slightly different behavior from corresponding Impala functions that are passed down to Impala. Only when **SQL_FUNCTIONS=ALL** can the SAS/ACCESS engine also pass these SAS SQL functions to Impala. Due to incompatibility in date and time functions between Impala and SAS, Impala might not process them correctly. Check your results to determine whether these functions are working as expected.
Passing Joins to Impala

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for Impala

**Loading**

Bulk loading with the Impala engine is accomplished in two ways:

- Use the WebHDFS interface to Hadoop to push data to HDFS. The SAS environment variable `SAS_HADOOP_RESTFUL` must be defined and set to the value of 1. You can include the properties for the WebHDFS location in the Hadoop `hdfs-site.xml` file. Alternatively, specify the WebHDFS host name or the IP address of the server where the external file is stored using the `BL_HOST=` option. The `BULKLOAD=` option must be set to YES. No JAR files are needed.

  *Note:* Support for the `SAS_HADOOP_RESTFUL` environment variable was added in the second maintenance release for SAS 9.4.

- Configure a required set of Hadoop JAR files. JAR files must be in a single location and available to the SAS client machine. The SAS environment variable `SAS_HADOOP_JAR_PATH` must be defined and set to the location of the Hadoop JAR files. Refer to the `SAS 9.4 Hadoop Configuration Guide for Base SAS and SAS/ACCESS` for information about what Cloudera JAR files are required.

  *Note:* Support for configuration using Hadoop JAR files and the `SAS_HADOOP_JAR_PATH` environment variable was added in the second maintenance release for SAS 9.4.

Here is how the Impala engine creates table data using the bulk-loading process:

1. SAS issues two `CREATE TABLE` statements to the Impala server. One `CREATE TABLE` statement creates the target Impala table. The other `CREATE TABLE` statement creates a temporary table.

2. SAS uses WebHDFS to upload table data to the HDFS `/tmp` directory. The resulting file is a UTF-8 delimited text file.

3. SAS issues a `LOAD DATA` statement to move the data file from the HDFS `/tmp` directory into the temporary table.

4. SAS issues an `INSERT INTO` statement that copies and transforms the text data from the temporary table into the target Impala table.

5. SAS deletes the temporary table.
Data Set Options for Bulk Loading

Here are the Impala bulk-loading data set options. The BULKLOAD= data set option is required for bulk loading, and all others are optional. For more information, see Chapter 12, “Data Set Options for Relational Databases,” on page 237.

- BL_DATAFILE=
- BL_DELETE_DATAFILE=
- BL_HOST=
- BL_PORT=
- BULKLOAD=
- CONFIG=
- CONFIGDIR=
- HDFS_PRINCIPAL=

Example 1: Create an Impala Table from a SAS Data Set

This example shows how you can use a SAS data set, SASFLT.FLT98, to create and load an Impala table, FLIGHTS98.

libname sasflt 'SAS-data-library';
libname mydblib impala host=mysrv1
  db=users user=myusr1 password=mypwd1;
proc sql;
create table mydblib.flights98
  (BULKLOAD=YES
   BL_DATAFILE='/tmp/mytable.dat'
   BL_HOST='192.168.x.x'
   BL_PORT=50070)
  as select * from sasflt.flt98;
quit;

Example 2: Append a SAS Data Set to an Existing Impala Table

This example shows how you can append the SAS data set, SASFLT.FLT98, to the existing Impala table, FLIGHTS98. In this example, the HDFS_PRINCIPAL data set option is specified as well. You specify the HDFS_PRINCIPAL= data set option when you configure HDFS to allow Kerberos authentication. Rather than deleting the data file, BL_DELETE_DATAFILE=NO causes the engine to leave it after the load has completed.

proc append base=mydblib.flights98
  (BULKLOAD=yes
   BL_DATAFILE='/tmp/mytable.dat'
   BL_DELETE_DATAFILE=no
   HDFS_PRINCIPAL='hdfs/hdfs_host.example.com@test.example.com'
   BL_HOST='192.168.x.x'
   BL_PORT=50070)
data=sasflt.flt98;
Naming Conventions for Impala

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The Impala interface supports table names and column names that contain up to 32 characters. If column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name would result in identical column names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a name longer than 32 characters. If you have a table name that is greater than 32 characters, it is recommended that you create a table view.

Although the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options are supported for SAS/ACCESS Interface to Impala, you should not need to use them. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Impala is not case sensitive, so all names default to lowercase.

SAS and Impala objects include tables, views, table references, and columns. They follow these naming conventions.

• A SAS name must be from 1 to 32 characters long. When Impala column names and table names are 32 characters or less, SAS handles them seamlessly. When SAS reads Impala column names that are longer than 32 characters, a generated SAS variable name is truncated to 32 characters. Impala table names should be 32 characters or less because SAS cannot truncate a table reference. If you already have a table name that is greater than 32 characters, create an Impala table view or use the explicit SQL feature of PROC SQL to access the table.

• If truncating would result in identical names, SAS generates a unique name.

• Even when it is enclosed in single or double quotation marks, an Impala name does not retain case sensitivity. Impala table and column names can contain uppercase letters A through Z (A–Z), lowercase letters A through Z (a–z), numbers from 0 to 9, and the underscore (_). Impala converts uppercase characters to lowercase. Therefore, such SAS table references as MYTAB and mytab are synonyms that refer to the same table.

• A name can begin with a letter but not an underscore or a number.

• A name cannot be an Impala reserved word. If a name generates an Impala error, try to append a number or underscore in an appropriate place. For example, if shipped results in an error, try shipped1 or ship_date.

Although the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options are supported for SAS/ACCESS Interface to Impala, you should not need to use them. (For information about these options, see LIBNAME Option for Relational Databases on page 96.)
Data Types for Impala

Overview

Every column in a table has a name and a data type. The data type tells Impala how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Impala data types, null and default values, and data conversions.

SAS/ACCESS Interface to Impala does not directly support data types that are not listed below. Any columns that use unsupported data types are read into SAS as character strings.

For more information about Impala data types and to determine which data types are available for your version of Impala, see your Impala documentation.

Supported Impala Data Types

Here are the data types that the Impala engine supports.

- Numeric data:
  
  - BIGINT
  - FLOAT
  - TIMESTAMP
  - INT
  - BOOLEAN
  - SMALLINT
  - DOUBLE
  - TINYINT

  Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

- Character data:
  
  - CHARn
  - VARCHARn
  - STRING

- Date and time data: TIMESTAMP

  Note: In the third maintenance release for SAS 9.4, support for CHAR and VARCHAR data types was added.

To use CHAR and VARCHAR, an Impala 2.0 server and the Cloudera 2.5.22 or higher ODBC client driver are required.

SAS Data Types

SAS has two fundamental data types, character and numeric. SAS character variables (columns) are of a fixed length with a maximum of 32,767 characters. SAS numeric
variables are signed eight-byte, floating-point numbers. When SAS numerics are used in conjunction with SAS formats, they can represent a number of data types, including DATE, TIME, and DATETIME. For more information about SAS data types, see *SAS Language Reference: Concepts*.

**Data Conversion from Impala to SAS**

This table shows the default SAS formats that are assigned to SAS variables that are created when SAS/ACCESS reads Impala table columns.

*Note:* In the third maintenance release for SAS 9.4, support for CHAR and VARCHAR data types was added.

**Table 21.3  Impala to SAS: Default SAS Formats for Impala Data Types**

<table>
<thead>
<tr>
<th>Impala Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>character</td>
<td>$255.</td>
</tr>
<tr>
<td>STRING</td>
<td></td>
<td>$32767.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td></td>
<td>$65355</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>numeric</td>
<td>1.</td>
</tr>
<tr>
<td>BIGINT</td>
<td></td>
<td>20.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>INT</td>
<td></td>
<td>11.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td></td>
<td>6.</td>
</tr>
<tr>
<td>TINYINT</td>
<td></td>
<td>4.</td>
</tr>
</tbody>
</table>

**Issues When Converting from Impala to SAS**

Below are some issues that you might face when you convert from Impala to SAS:

- **STRING:** Depending on the length of Impala string data, the SAS character format ($32767.$) might be unnecessarily large for short STRING columns. Alternatively, the SAS character format ($32767.$) might truncate Impala STRING columns that contain more than 32,767 characters.

- **BIGINT:** Converting an Impala BIGINT to a SAS numeric can result in loss of precision because the internal SAS eight-byte, floating-point format accurately preserves only 15 digits of precision. An Impala BIGINT preserves up to 19 digits of precision.
Data Conversion from SAS to Impala

This table shows the Impala data types that are assigned when SAS/ACCESS Interface to Impala creates an Impala table.

**Table 21.4  SAS to Impala: Default Impala Data Types for SAS Formats**

<table>
<thead>
<tr>
<th>SAS Data Type</th>
<th>SAS Format</th>
<th>Impala Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>$n.</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td></td>
<td>$n.</td>
<td>STRING</td>
</tr>
<tr>
<td></td>
<td>$n.</td>
<td>VARCHAR(n)</td>
</tr>
<tr>
<td>numeric</td>
<td>DATETIMEw.p</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>DATEw.</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>TIMEw.</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>1. to 2.</td>
<td>TINYINT</td>
</tr>
<tr>
<td></td>
<td>3. to 4.</td>
<td>SMALLINT</td>
</tr>
<tr>
<td></td>
<td>5. to 9.</td>
<td>INT</td>
</tr>
<tr>
<td></td>
<td>10. to 18.</td>
<td>BIGINT</td>
</tr>
<tr>
<td></td>
<td>other numeric formats</td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>

Impala Null Values

Impala has a special value called NULL. An Impala NULL value represents an absence of information and is analogous to a SAS missing value. When SAS/ACCESS Interface to Impala reads an Impala NULL value, it interprets it as a SAS missing value. For more information about how SAS handles NULL values, see
Chapter 22
SAS/ACCESS Interface to Informix

Introduction to SAS/ACCESS Interface to Informix
  Overview ................................................. 664
  Default Environment .................................. 664

LIBNAME Statement Specifics for Informix
  Overview ................................................. 664
  Arguments ............................................... 664
  Informix LIBNAME Statement Example ............. 667

Data Set Options for Informix ......................... 667

SQL Pass-Through Facility Specifics for Informix
  Key Information ....................................... 668
  Stored Procedures and the SQL Pass-Through Facility ..... 669
  Command Restrictions for the SQL Pass-Through Facility ................................................. 669
  Examples ............................................... 669

Autopartitioning Scheme for Informix ................. 671
  Overview ................................................. 671
  Autopartitioning Restrictions ..................... 671
  Using WHERE Clauses .................................. 671
  Using DBSLICEPARM= .................................. 671
  Using DBSLICE= ....................................... 672

Temporary Table Support for Informix ................. 672

Passing SAS Functions to Informix .................... 672

Passing Joins to Informix ................................ 673

Locking in the Informix Interface ...................... 673

Naming Conventions for Informix ....................... 674

Data Types for Informix ................................ 674
  Overview ................................................. 674
  Supported Informix Data Types .................... 675
  LIBNAME Statement Data Conversions ............... 676
  SQL Pass-Through Facility Data Conversions ........ 677

Informix Servers ........................................ 677
  Overview: Informix Database Servers ............... 677
  Using the DBDATASRC Environment Variable ........ 678
  Using Fully Qualified Table Names ................. 678
Introduction to SAS/ACCESS Interface to Informix

Overview

For available SAS/ACCESS features, see Informix supported features on page 82. For background information about Informix, see Informix servers on page 677. For more information about Informix, see your Informix documentation.

Default Environment

When you access Informix tables by using SAS/ACCESS Interface to Informix, the default Informix read isolation level is set for committed reads, and SAS spooling is on. Committed reads enable you to read rows unless another user or process is updating the rows. Reading in this manner does not lock the rows. SAS spooling guarantees that you obtain identical data each time you re-read a row because SAS buffers the rows after you read them the first time. This default environment is suitable for most users. If this default environment is unsuitable for your needs, see “Locking in the Informix Interface” on page 673.

To see the SQL statements that SAS issues to the Informix server, include the SASTRACE= option in your code:

```
option sastrace=',,,d';
```

If you use quotation marks in your Informix SQL statements, set your DELIMIDENT= environment variable to DELIMIDENT=YES or Informix might reject your statements. Because some SAS options that preserve case generate SQL statements that contain quotation marks, you should set DELIMIDENT=YES in your environment.

LIBNAME Statement Specifics for Informix

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Informix supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Informix.

```
LIBNAME libref informix <connection-options> <LIBNAME-options>;
```

Arguments

- `libref` specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.
- `informix` specifies the SAS/ACCESS engine name for the Informix interface.
connection-options

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. Here is how these options are defined.

USER=<'Informix-user-name'><

specifies the Informix user name that you use to connect to the database that contains the tables and views that you want to access. If you omit the USER= option, your operating environment account name is used, if applicable to your operating environment.

USING=<'Informix-password'><

specifies the password that is associated with the Informix user. If you omit the password, Informix uses the password in the /etc/password file.

Alias: PASSWORD=, PWD=

SERVER=<'ODBC-data-source'><

specifies the ODBC data source to which you want to connect. An error occurs if the SERVER= option is not set. For UNIX platforms, you must configure the data source by modifying the odbc.ini file. See your ODBC driver documentation for details.

For the SAS/ACCESS 9 Interface to Informix, the Informix ODBC Driver API is used to connect to Informix, and connection options have changed accordingly. The DATABASE= option from the SAS 8 version of SAS/ACCESS was removed. If you need to specify a database, set it in the odbc.ini file. For SERVER= options, instead of specifying the server name, as in SAS 8, specify an ODBC data source name. You can also use a user ID and password with SERVER=.

DBDATASRC=<'database-data-source'><

environment variable that lets you set a default data source. This value is used if you do not specify a SERVER= connection option.

LIBNAME-options

define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Informix, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

Table 22.1  SAS/ACCESS LIBNAME Options for Informix

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>YES</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 when inserting rows; 0 when updating rows</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DLBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DLBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>NO</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS, 2 or 3</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>LOCKTABLE=</td>
<td>no locking</td>
</tr>
<tr>
<td>LOCKTIME=</td>
<td>none</td>
</tr>
<tr>
<td>LOCKWAIT=</td>
<td>not set</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>NONE</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>NO</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>NO</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>COMMITTED READ (see “Locking in the Informix Interface”)</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>your user name</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
</tbody>
</table>
Informix LIBNAME Statement Example

In this example, the libref MYDBLIB uses the Informix interface to connect to an Informix database:

```
libname mydblib informix user=myusr1 using=mypwd1 server=mysrv1;
```

In this example USER=, USING=, and SERVER= are connection options.

Data Set Options for Informix

All SAS/ACCESS data set options in this table are supported for Informix. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

Table 22.2 SAS/ACCESS Data Set Options for Informix

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULL=</td>
<td><em>ALL</em>=YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for Informix”</td>
</tr>
</tbody>
</table>
Here are the SQL pass-through facility specifics for the Informix interface.

- The `dbms-name` is `informix`.
- The CONNECT statement is optional when you are connecting to an Informix database if the DBDATASRC environment variable has been set. When you omit a CONNECT statement, an implicit connection is performed when the first EXECUTE statement or CONNECTION TO component is passed to the DBMS.
- You can connect to only one Informix database at a time. However, you can specify multiple CONNECT statements if they all connect to the same Informix database. If you use multiple connections, you must use an alias to identify the different connections. If you omit an alias, `informix` is automatically used.
- The CONNECT statement `database-connection-arguments` are identical to its LIBNAME connection options.
- If you use quotation marks in your Informix pass-through statements, your DELIMIDENT= environment variable must be set to DELIMIDENT=YES, or your statements are rejected by Informix.
- The SCHEMA LIBNAME option is ignored when using implicit pass-through.
- The SCHEMA data set option disables implicit pass-through.

### SQL Pass-Through Facility Specifics for Informix

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS, 2 or 3</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>see “Data Types for Informix”</td>
</tr>
<tr>
<td>ERLLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>LOCKTABLE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>DATETIME</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>
Stored Procedures and the SQL Pass-Through Facility

The SQL pass-through facility recognizes two types of stored procedures in Informix that perform only database functions. The methods for executing the two types of stored procedures are different.

Procedures that return no values to the calling application

Stored procedures that do not return values can be executed directly by using the Informix SQL EXECUTE statement. Stored procedure execution is initiated with the Informix EXECUTE PROCEDURE statement. The following example executes the stored procedure `make_table`. The stored procedure has no input parameters and returns no values.

```sql
execute (execute procedure make_table())
    by informix;
```

Procedures that return values to the calling application

Stored procedures that return values must be executed by using the PROC SQL SELECT statement with a CONNECTION TO component. This example executes the stored procedure `read_address`, which has one parameter, "Putnum".

The values that `read_address` returns serve as the contents of a virtual table for the PROC SQL SELECT statement.

```sql
select * from connection to informix
    (execute procedure read_address ("Putnum"));
```

For example, when you try to execute a stored procedure that returns values from a PROC SQL EXECUTE statement, you receive this error message:

```sql
execute (execute procedure read_address
    ("Putnum")) by informix;
```

ERROR: Informix EXECUTE Error: Procedure
(read_address) returns too many values.

Command Restrictions for the SQL Pass-Through Facility

Informix SQL contains extensions to the ANSI-89 standards. Some of these extensions, such as LOAD FROM and UNLOAD TO, are restricted from use by any applications other than the Informix DB-Access product. Specifying these extensions in the PROC SQL EXECUTE statement generates this error:

-201
A syntax error has occurred

Examples

This example connects to Informix by using data source `mysrv1`:

```sql
proc sql;
    connect to informix
        (user=myusr1 password=mypwd1 server=mysrv1);
```

You can use the DBDATASRC environment variable to set the default data source.

This next example grants UPDATE and INSERT authority to user `gomez` on the Informix ORDERS table. Because the CONNECT statement is omitted, an implicit
connection is made. The connection uses a default value of `informix` as the connection alias and default values for the `SERVER=` argument.

```sql
proc sql;
   execute (grant update, insert on ORDERS to gomez) by informix;
quit;
```

This example connects to Informix and drops (removes) the table TempData from the database. The alias Temp5 that is specified in the CONNECT statement is used in the EXECUTE statement's BY clause.

```sql
proc sql;
   connect to informix as temp5
       (server=mysrv1);
   execute (drop table tempdata) by temp5;
   disconnect from temp5;
quit;
```

This example sends an SQL query, shown with highlighting, to the database for processing. The results from the SQL query serve as a virtual table for the PROC SQL FROM clause. In this example `DBCON` is a connection alias.

```sql
proc sql;
   connect to informix as dbcon
       (user=myusr1 using=mypwd1
        server=mysrv1);
   select *
       from connection to dbcon
       (select empid, lastname, firstname,
        hiredate, salary
        from employees
        where hiredate>='31JAN88');
   disconnect from dbcon;
quit;
```

This next example gives the previous query a name and stores it as the PROC SQL view `Samples.Hires88`. The CREATE VIEW statement appears in highlighting.

```sql
libname samples 'SAS-library';
proc sql;
   connect to informix as mycon
       (user=myusr1 using=mypwd1
        server=mysrv1);
   create view samples.hires88 as
      select *
       from connection to mycon
       (select empid, lastname, firstname,
        hiredate, salary
        from employees
        where hiredate>='31JAN88');
   disconnect from mycon;
quit;
```

This example connects to Informix and executes the stored procedure `testproc`. The `select *` clause displays the results from the stored procedure.
proc sql;
   connect to informix as mydb
       (server=mysrv1);
   select * from connection to mydb
       (execute procedure testproc('123456'));
   disconnect from mydb;
quit;

Autopartitioning Scheme for Informix

Overview
Autopartitioning for SAS/ACCESS Interface to Informix is a modulo (MOD) function method. For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

Autopartitioning Restrictions
SAS/ACCESS Interface to Informix places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here is how columns are partitioned.

• INTEGER
• SMALLINT
• BIT
• TINYINT
• You can also use DECIMALS with 0-scale columns as the partitioning column.
• Nullable columns are the least preferable.

Using WHERE Clauses
Autopartitioning does not select a column to be the partitioning column if it appears in a SAS WHERE clause. For example, the following DATA step cannot use a threaded Read to retrieve the data because all numeric columns in the table are in the WHERE clause:

    data work.locemp;
    set trlib.MYEMPS;
    where EMPNUM<=30 and ISTENURE=0 and
         SALARY<=35000 and NUMCLASS>2;
    run;

Using DBSLICEPARM=
SAS/ACCESS Interface to Informix defaults to three threads when you use autopartitioning. However, do not specify a maximum number of threads in DBSLICEPARM= LIBNAME option on page 151 to use for the threaded Read.

This example shows how to use DBSLICEPARM= with the maximum number of threads set to five:
libname x informix user=myusr1 using=mypwd1 server=mysrv1;
proc print data=x.dept(dbsliceparm=(ALL,5));
run;

Using DBSLICE=

You can achieve the best possible performance when using threaded Reads by specifying the DBSLICE= data set option for Informix in your SAS operation. This example shows how to use it.

libname x informix user=myusr1 using=mypwd1 server=mysrv1;
data xottest;
set x.invoice(dbslice=('amtbilled<10000000' 'amtbilled>=10000000'));
run;

Temporary Table Support for Informix

SAS/ACCESS Interface to Informix supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

Passing SAS Functions to Informix

SAS/ACCESS Interface to Informix passes the following SAS functions to Informix for processing if the DBMS driver or client that you are using supports this function. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Maximum</td>
</tr>
<tr>
<td>ARCOS</td>
<td>MDY</td>
</tr>
<tr>
<td>ARSIN</td>
<td>MIN</td>
</tr>
<tr>
<td>ATAN</td>
<td>MINUTE</td>
</tr>
<tr>
<td>ATAN2</td>
<td>MONTH</td>
</tr>
<tr>
<td>AVG</td>
<td>SECOND</td>
</tr>
<tr>
<td>COS</td>
<td>SIN</td>
</tr>
<tr>
<td>COUNT</td>
<td>SQRT</td>
</tr>
<tr>
<td>DATE</td>
<td>STRIP</td>
</tr>
<tr>
<td>DAY</td>
<td>SUM</td>
</tr>
<tr>
<td>EXP</td>
<td>TAN</td>
</tr>
<tr>
<td>HOUR</td>
<td>TODAY</td>
</tr>
<tr>
<td>INT</td>
<td>WEEKDAY</td>
</tr>
<tr>
<td>LOG</td>
<td>YEAR</td>
</tr>
<tr>
<td>LOG10</td>
<td></td>
</tr>
</tbody>
</table>

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Informix. Due to incompatibility in date and time functions between
Informix and SAS, the Informix server might not process them correctly. Check your results to determine whether these functions are working as expected.

DATEPART  TIMEPART

---

**Passing Joins to Informix**

For a multiple libref join to pass to Informix, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (USING=)
- server (SERVER=)

Due to an Informix database limitation, the maximum number of tables that you can specify to perform a join is 22. An error message appears if you specify more than 22.

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

---

**Locking in the Informix Interface**

In most cases, SAS spooling is on by default for the Informix interface and provides the data consistency that you need.

To control how the Informix interface handles locking, you can use the `READ_ISOLATION_LEVEL= LIBNAME` option. Here are the valid values.

- **COMMITTED_READ**
  retrieves only committed rows. No locks are acquired, and rows can be locked exclusively for update by other users or processes. This is the default setting.

- **REPEATABLE_READ**
  gives you a shared lock on every row that is selected during the transaction. Other users or processes can also acquire a shared lock, but no other process can modify any row that is selected by your transaction. If you repeat the query during the transaction, you re-read the same information. The shared locks are released only when the transaction commits or rolls back. Another process cannot update or delete a row that is accessed by using a repeatable read.

- **DIRTY_READ**
  retrieves committed and uncommitted rows that might include phantom rows, which are rows that are created or modified by another user or process that might subsequently be rolled back. This type of read is most appropriate for tables that are not frequently updated.

- **CURSOR_STABILITY**
  gives you a shared lock on the selected row. Another user or process can acquire a shared lock on the same row, but no process can acquire an exclusive lock to modify data in the row. When you retrieve another row or close the cursor, the shared lock is released.

If you set `READ_ISOLATION_LEVEL= REPEATABLE_READ` or `CURSOR_STABILITY`, it is recommended that you assign a separate libref and clear
that libref when you finish working with the tables. This technique minimizes the negative performance impact on other users that occurs when you lock the tables. To clear the libref, include this code:

```sas
libname libref clear;
```

For current Informix releases, `READ_ISOLATION_LEVEL=` is valid only when transaction logging is enabled. If transaction logging is not enabled, an error is generated when you use this option. Also, locks placed when `READ_ISOLATION_LEVEL=REPEATABLE READ` or `CURSOR_STABILITY` are not freed until the libref is cleared.

To see the SQL locking statements that SAS issues to the Informix server, include in your code the `SASTRACE=` system option.

```sas
option sastrace=',,,d';
```

For more details about Informix locking, see your Informix documentation.

### Naming Conventions for Informix

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

The `PRESERVE_TAB_NAMES=` and `PRESERVE_COL_NAMES=` options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Informix is not case sensitive, so all names default to lowercase.

Informix objects include tables and columns. They follow these naming conventions.

- Although table and column names must be from 1 to 32 characters, the limitation on some Informix servers might be lower.
- Table and column names must begin with a letter or an underscore (\_) that is followed by letters, numbers, or underscores. Special characters are not supported. However, if you enclose a name in quotation marks and `PRESERVE_TAB_NAMES=YES` (when applicable), it can begin with any character.

Because several problems were found in the Informix ODBC driver that result from using uppercase or mixed case, Informix encourages users to use lowercase for table and column names. Informix currently has no schedule for fixing these known problems.

### Data Types for Informix

#### Overview

Every column in a table has a name and a data type. The data type tells Informix how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Informix data types, null values, and data conversions.
**Supported Informix Data Types**

**Data Types**
Here are the data types that the Informix engine supports.

- **Character data:**
  - `CHAR(n)`
  - `NVARCHAR(m,n)`
  - `NCHAR(n)`
  - `TEXT`
  - `VARCHAR(m,n)`
  - `BYTE`

- **Numeric data:**
  - `DECIMAL`
  - `REAL`
  - `MONEY`
  - `SMALLFLOAT`
  - `NUMERIC`
  - `SERIAL`
  - `FLOAT`
  - `SMALLINT`
  - `DOUBLE PRECISION`
  - `INT8`
  - `INTEGER`
  - `SERIAL8`

*Note:* When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

**TIP** When the length value of INT8 or SERIAL8 is greater than 15, the last few digits currently do not display correctly due to a display limitation.

- **Date, time, and interval data:**
  - `DATE`
  - `INTERVAL`
  - `DATETIME`

**Informix Null Values**
Informix has a special value that is called NULL. An Informix NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads an Informix NULL value, it interprets it as a SAS missing value.

If you do not indicate a default value for an Informix column, the default value is NULL. You can specify the keywords NOT NULL after the data type of the column when you create an Informix table to prevent NULL values from being stored in the column. When you create an Informix table with SAS/ACCESS, you can use the `DBNULL=` data set option to indicate whether NULL is a valid value for specified columns.

For more information about how SAS handles NULL values, see Potential Result Set Differences When Processing Null Data on page 33.

To control how the DBMS handles SAS missing character values, use the `NULLCHAR=` and `NULLCHARVAL=` data set options.
**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to Informix assigns to SAS variables when using the LIBNAME statement to read from an Informix table. These default formats are based on Informix column attributes. To override these default data types, use the `DBTYPE=` data set option on a specific data set.

**Table 22.3  LIBNAME Statement: Default SAS Formats for Informix Data Types**

<table>
<thead>
<tr>
<th>Informix Column Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)</td>
<td>$n</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE9.</td>
</tr>
<tr>
<td>DATETIME***</td>
<td>DATETIME24.5</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>m+2.n</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT</td>
<td>none</td>
</tr>
<tr>
<td>INTEGER</td>
<td>none</td>
</tr>
<tr>
<td>INT8**</td>
<td>none</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>$n</td>
</tr>
<tr>
<td>MONEY</td>
<td>none</td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td>$n</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>NVARCHAR(m,n)*</td>
<td>$m</td>
</tr>
<tr>
<td>REAL</td>
<td>none</td>
</tr>
<tr>
<td>SERIAL</td>
<td>none</td>
</tr>
<tr>
<td>SERIAL8**</td>
<td>none</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>none</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>none</td>
</tr>
<tr>
<td>TEXT*</td>
<td>$n</td>
</tr>
</tbody>
</table>
Informix Column Type | Default SAS Format
---|---
VARCHAR\((m,n)\)* | $m$

* Only supported by Informix online databases.

** The precision of an INT8 or SERIAL8 is 15 digits.

*** If the Informix field qualifier specifies either HOUR, MINUTE, SECOND, or FRACTION as the largest unit, the value is converted to a SAS TIME value. All other values (such as YEAR, MONTH, or DAY) are converted to a SAS DATETIME value.

The following table shows the default Informix data types that SAS/ACCESS applies to SAS variable formats during output operations when you use the LIBNAME statement.

**Table 22.4 LIBNAME Statement: Default Informix Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Informix Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.$</td>
<td>CHAR(w).</td>
</tr>
<tr>
<td>(w). with SAS format name of NULL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(w.d). with SAS format name of NULL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>all other numerics</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>datetime(w.d)</td>
<td>DATETIME YEAR TO FRACTION(5)</td>
</tr>
<tr>
<td>date(w).</td>
<td>DATE</td>
</tr>
<tr>
<td>time.(w).</td>
<td>DATETIME HOUR TO SECOND</td>
</tr>
</tbody>
</table>

**SQL Pass-Through Facility Data Conversions**

The SQL pass-through facility uses the same default conversion formats as the LIBNAME statement. For conversion tables, see “LIBNAME Statement Data Conversions” on page 676.

**Informix Servers**

**Overview: Informix Database Servers**

There are two types of Informix database servers, the Informix OnLine and Informix SE servers. Informix OnLine database servers can support many users and provide tools that ensure high availability, high reliability, and that support critical applications. Informix SE database servers are designed to manage relatively small databases that individuals use privately or that a small number of users share.
Using the DBDATASRC Environment Variable

The SQL pass-through facility supports the DBDATASRC environment variable, which is an extension to the Informix environment variable. If you set DBDATASRC, you can omit the CONNECT statement. The value of DBDATASRC is used instead of the SERVER= argument in the CONNECT statement. The syntax for setting DBDATASRC is like the syntax of the SERVER= argument:

Bourne shell:

```plaintext
export DBDATABASE='mysrv1'
```

C shell:

```plaintext
setenv DBDATASRC mysrv1
```

If you set DBDATASRC, you can issue a PROC SQL SELECT or EXECUTE statement without first connecting to Informix with the CONNECT statement.

If you omit the CONNECT statement, an implicit connection is performed when the SELECT or EXECUTE statement is passed to Informix.

If you create an SQL view without an explicit CONNECT statement, the view can dynamically connect to different databases, depending on the value of the DBDATASRC environment variable.

Using Fully Qualified Table Names

Informix supports a connection to only one database. If you have data that spans multiple databases, you must use fully qualified table names to work within the Informix single-connection constraints.

In this example, the tables Tab1 and Tab2 reside in different databases, MyDB1 and MyDB2, respectively.

```plaintext
proc sql;
  connect to informix
    (server=mysrv1);

  create view tab1v as
    select * from connection
    to informix
    (select * from mydb1.tab1);

  create view tab2v as
    select * from connection
    to informix
    (select * from mydb2.tab2);

quit;

data getboth;
  merge tab1v tab2v;
  by common;
run;
```

Because the tables reside in separate databases, you cannot connect to each database with a PROC SQL CONNECT statement and then retrieve the data in a single step. Using the fully qualified table name (that is, `database.table`) enables you to use any
Informix database in the CONNECT statement and access Informix tables in the same or different databases in a single SAS procedure or DATA step.
Chapter 23
SAS/ACCESS Interface to Microsoft SQL Server

Introduction to SAS/ACCESS Interface to Microsoft SQL Server .................. 681
LIBNAME Statement Specifics for Microsoft SQL Server .......................... 682
Overview ......................................................... 682
Arguments ...................................................... 682
Microsoft SQL Server LIBNAME Statement Examples .......................... 686
Data Set Options for Microsoft SQL Server ........................................... 686
SQL Pass-Through Facility Specifics for Microsoft SQL Server ................. 688
Key Information .................................................. 688
CONNECT Statement Examples ......................................................... 689
Connection to Component Examples ................................................... 689
DBLOAD Procedure Specifics for Microsoft SQL Server ......................... 690
Overview ......................................................... 690
Examples ....................................................... 690
Passing SAS Functions to Microsoft SQL Server .................................. 691
Locking in the Microsoft SQL Server Interface .................................. 691
Naming Conventions for Microsoft SQL Server .................................. 693
Data Types for Microsoft SQL Server ............................................... 693
Overview ......................................................... 693
Microsoft SQL Server Null Values .................................................... 693
LIBNAME Statement Data Conversions .............................................. 693

Introduction to SAS/ACCESS Interface to Microsoft SQL Server

For available SAS/ACCESS features, see Microsoft SQL Server supported features. For more information about Microsoft SQL Server, see your Microsoft SQL Server documentation.

SAS/ACCESS Interface to Microsoft SQL Server has been tested and certified against Data Direct Technologies Connect ODBC and Data Direct SequeLink ODBC products.
LIBNAME Statement Specifics for Microsoft SQL Server

Overview

This section describes the LIBNAME statement as supported in SAS/ACCESS Interface to Microsoft SQL Server. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Microsoft SQL Server.

LIBNAME libref sqlsvr <connection-options> <LIBNAME-options>;

Arguments

libref
specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

sqlsvr
specifies the SAS/ACCESS engine name for the Microsoft SQL Server interface.

collection-options
provide connection information and control how SAS manages the timing and concurrency of the connection to the DBMS. When you use the LIBNAME statement, you can connect to Microsoft SQL Server in many different ways. Specify only one of these methods for each connection because they are mutually exclusive.

- USER=, PASSWORD=, and DATASRC=
- COMPLETE=
- NOPROMPT=
- PROMPT=
- REQUIRED=

Here is how these options are defined.

USER=&user-name&
lets you connect to Microsoft SQL Server with a user ID that is different from the default ID.

Alias: UID=

Usage: optional

PASSWORD=&password&
specifies the Microsoft SQL Server password that is associated with your user ID.

Alias: PWD=

Usage: optional

DATASRC=&SQL-Server-data-source&
specifies the Microsoft SQL Server data source to which you want to connect. For UNIX platforms, data sources must be configured by modifying
the .ODBC.ini file. DSN= indicates that the connection is attempted using the ODBC SQLConnect API, which requires a data source name. You can also use a user ID and password with DSN=. This API is guaranteed to be present in all drivers.

Alias: DATABASE=, DB=, DSN=

**COMPLETE=**<>', SQL-Server-connection-options<>

specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMS macro variable. If you do not specify enough correct connection options, you are prompted with a dialog box that displays the values from the COMPLETE= connection string. You can edit any field before you connect to the data source. See your driver documentation for more details.

**NOPROMPT=**<>', SQL-Server-connection-options<>

specifies connection options for your data source or database. Separate multiple options with a semicolon. If you do not specify enough correct connection options, an error is returned. No dialog box is displayed to help you with the connection string.

**PROMPT=**<>', SQL-Server-connection-options<>

specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMS macro variable. PROMPT= does not immediately try to connect to the DBMS. Instead, it displays a dialog box that contains the values that you entered in the PROMPT= connection string. You can edit values or enter additional values in any field before you connect to the data source.

**REQUIRED=**<>', SQL-Server-connection-options<>

specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMS macro variable. If you do not specify enough correct connection options, a dialog box prompts you for the connection options. REQUIRED= lets you modify only required fields in the dialog box.

These Microsoft SQL Server connection options are not supported on UNIX.

- **BULKCOPY=**
- **COMPLETE=**
- **PROMPT=**
- **REQUIRED=**

**LIBNAME-options**

define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Microsoft SQL Server, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>varies with transaction type</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>UNIQUE when data source supports only one cursor per connection; otherwise, SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>DYNAMIC</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DELETE_MUTL_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>1</td>
</tr>
<tr>
<td>KEYSET_SIZE=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>NONE</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for Microsoft SQL Server”</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>see “Naming Conventions for Microsoft SQL Server”</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>none</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>0</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the Microsoft SQL Server Interface”)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the Microsoft SQL Server Interface”)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE_SQL=</td>
<td>driver-specific</td>
</tr>
<tr>
<td>USE_ODBC_CL=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>
**Microsoft SQL Server LIBNAME Statement Examples**

In this example, USER= and PASSWORD= are connection options.

```sql
libname mydblib sqlsvr user=myusr1 password=mypwd1;
```

In this next example, the libref MYDBLIB connects to a Microsoft SQL Server database using the NOPROMPT= option.

```sql
libname mydblib sqlsvr noprompt="uid=myusr1; pwd=mypwd1; dsn=sqlservr;" stringdates=yes;
```

```sas
proc print data=mydblib.customers;
  where state='CA';
run;
```

---

**Data Set Options for Microsoft SQL Server**

All SAS/ACCESS data set options in this table are supported for Microsoft SQL Server. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

**Table 23.2 SAS/ACCESS Data Set Options for Microsoft SQL Server**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURSOR_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DUBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBASTYPE=</td>
<td>see “Data Types for Microsoft SQL Server”</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS, 2 or 3</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for Microsoft SQL Server”</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT_SQL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>KEYSET_SIZE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for Microsoft SQL Server

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Microsoft SQL Server interface under UNIX hosts.

- The `dbms-name` is `SQLSVR`.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to Microsoft SQL Server. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default alias is used. The functionality of multiple connections to the same Microsoft SQL Server data source might be limited by the particular data source driver.
- The CONNECT statement `database-connection-arguments` are identical to its LIBNAME connection options.
- These LIBNAME options are available with the CONNECT statement:
  - `AUTOCOMMIT=`
  - `CURSOR_TYPE=`
  - `KEYSET_SIZE=`
  - `QUERY_TIMEOUT=`
  - `READBUFF=`
  - `READ_ISOLATION_LEVEL=`
  - `TRACE=`
  - `TRACEFILE=`
  - `USE_ODBC_CL=`
- The `DBMS-SQL-query` argument can be a DBMS-specific SQL EXECUTE statement that executes a DBMS stored procedure. However, if the stored procedure contains more than one query, only the first query is processed.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_SQL=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>


**CONNECT Statement Examples**

These examples connect to a data source that is configured under the data source name **User's Data** using the alias `USER1`. The first example uses the connection method that is guaranteed to be present at the lowest level of conformance. Note that `DATASRC=` names can contain quotation marks and spaces.

```sql
proc sql;
   connect to sqlsvr as user1
   (datasrc="User's Data" user=myusr1 password=mypwd1);
```

This example uses the connection method that represents a more advanced level of Microsoft SQL Server ODBC conformance. It uses the input dialog box that is provided by the driver. The `DSN=` and `UID=` arguments are within the connection string. The SQL pass-through facility therefore does not parse them but instead passes them to the ODBC driver manager.

```sql
proc sql;
   connect to SQLSVR as user1
   (required = "dsn=User's Data; uid=myusr1");
```

In this example, you can select any data source that is configured on your machine. The example uses the connection method that represents a more advanced level of Microsoft SQL Server ODBC conformance, Level 1. When connection succeeds, the connection string is returned in the `SQLXMSG` and `SYSDBMS` macro variables. It can then be stored if you use this method to configure a connection for later use.

```sql
proc sql;
   connect to SQLSVR (required);
```

This example prompts you to specify the information that is required to make a connection to the DBMS. You are prompted to supply the data source name, user ID, and password in the dialog boxes that are displayed.

```sql
proc sql;
   connect to SQLSVR (prompt);
```

**Connection to Component Examples**

This example sends Microsoft SQL Server 6.5 (configured under the data source name "SQL Server") an SQL query for processing. The results from the query serve as a virtual table for the `PROC SQL FROM` clause. In this example, `MYDB` is the connection alias.

```sql
proc sql;
   connect to SQLSVR as mydb
   (datasrc="SQL Server" user=myusr1 password=mypwd1);
   select * from connection to mydb
   (select CUSTOMER, NAME, COUNTRY
    from CUSTOMERS
    where COUNTRY <> 'USA');
quit;
```

This next example returns a list of the columns in the CUSTOMERS table.

```sql
proc sql;
   connect to SQLSVR as mydb
   (datasrc = "SQL Server" user=myusr1 password=mypwd1);
```
DBLOAD Procedure Specifics for Microsoft SQL Server

Overview

For general information about this feature, see the DBLOAD Procedure on page 985.

The Microsoft SQL Server under UNIX hosts interface supports all DBLOAD procedure statements (except ACCDESC=) in batch mode. Here are SAS/ACCESS Interface to Microsoft SQL Server specifics for the DBLOAD procedure.

- The DBLOAD step DBMS= value is SQLSVR.
- Here are the database description statements that PROC DBLOAD uses:

  DSN= '<database-name>';  
  specifies the name of the database in which you want to store the new Microsoft SQL Server table. The database-name is limited to eight characters.

  The database that you specify must already exist. If the database name contains the _, $, @, or # special character, you must enclose it in quotation marks. The Microsoft SQL Server standard recommends against using special characters in database names, however

  USER= '<user name>';  
  enables you to connect to a Microsoft SQL Server database with a user ID that is different from the default ID.

  USER= is optional in the Microsoft SQL Server interface. If you specify USER=, you must also specify PASSWORD=. If USER= is omitted, your default user ID is used.

  PASSWORD= '<password>';  
  specifies the Microsoft SQL Server password that is associated with your user ID.

  PASSWORD= is optional in the Microsoft SQL Server interface because users have default user IDs. If you specify USER=, you must specify PASSWORD=. If you do not wish to enter your SQL Server password in clear text on this statement, see PROC PWENCODE in the Base SAS Procedures Guide for a method to encode it.

Examples

This example creates a new Microsoft SQL Server table, MYUSR1.EXCHANGE, from the DLIB.RATEOFEX data file. You must be granted the appropriate privileges in order to create new Microsoft SQL Server tables or views.

```sas
proc dbload dbms=SQLSVR data=dlib.rateofex;  
dsn=sample; user='myusr1'; password='mypwd1'; table=exchange;  
rename fgnindol=fgnindollars  
4=dollarsinfgn;
```
nulls updated=n fgnindollars=n
dollarsinfgn=n country=n;
load;
run;

This example only sends a Microsoft SQL Server SQL GRANT statement to the
SAMPLE database and does not create a new table. Therefore, the TABLE= and LOAD
statements are omitted.

proc dbload dbms=SQLSVR;
  user='myusr1';
  password='mypwd1';
  dsn=sample;
  sql grant select on myusr1.exchange
to testcase;
run;

---

**Passing SAS Functions to Microsoft SQL Server**

SAS/ACCESS Interface to Microsoft SQL Server passes the following SAS functions to
the data source for processing if the DBMS server supports this function. For details, see
"Passing Functions to the DBMS Using PROC SQL" on page 44.

ABS  LOWCASE
ARCOS MAX
ARSIN MIN
ATAN SIGN
AVG  SIN
CEIL SQRT
COS  TAN
EXP  UPCASE
FLOOR SUM
LOG  COUNT
LOG10

---

**Locking in the Microsoft SQL Server Interface**

These LIBNAME and data set options let you control how the Microsoft SQL Server
interface handles locking. For details about options, see "LIBNAME Options for
Relational Databases" on page 101.

READ_LOCK_TYPE= ROW | TABLE | NOLOCK
UPDATE_LOCK_TYPE= ROW | TABLE | NOLOCK
READ_ISOLATION_LEVEL= S | RR | RC | RU | V

The Microsoft SQL Server ODBC driver manager supports the S, RR, RC, RU, and
V isolation levels, as defined in this table.
Table 23.3 Isolation Levels for Microsoft SQL Server

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (serializable)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads.</td>
</tr>
<tr>
<td>RR (repeatable read)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RC (read committed)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RU (read uncommitted)</td>
<td>Allows dirty Reads, nonrepeatable Reads, and phantom Reads.</td>
</tr>
<tr>
<td>V (versioning)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads. These transactions are serializable but higher concurrency is possible than with the serializable isolation level. Typically, a nonlocking protocol is used.</td>
</tr>
</tbody>
</table>

Here is how the terms in the table are defined.

Dirty read
A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, the transaction can see changes that are made by those concurrent transactions even before they commit.

For example, if transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.

Nonrepeatable read
If a transaction exhibits this phenomenon, it might read a row once and later fail when it attempts to read that row again in the same transaction. The row might have been changed or even deleted by a concurrent transaction. Therefore, the read is not necessarily repeatable.

For example, if transaction T1 retrieves a row, transaction T2 updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.

Phantom reads
When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

For example, transaction T1 retrieves the set of all rows that satisfy some condition. If transaction T2 inserts a new row that satisfies that same condition and transaction T1 repeats its retrieval request, it sees a row that did not previously exist, a phantom.

UPDATE_ISOLATION_LEVEL= S | RR | RC | V
The Microsoft SQL Server ODBC driver manager supports the S, RR, RC, and V isolation levels that are defined in the preceding table.
Naming Conventions for Microsoft SQL Server

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Microsoft SQL Server is not case sensitive, so all names default to mixed case.

Microsoft SQL Server supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, SAS truncates them to 32 characters. If truncating a column name would result in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

Data Types for Microsoft SQL Server

Overview

Every column in a table has a name and a data type. The data type tells the Microsoft SQL Server how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Microsoft SQL Server null and default values and data conversions.

Microsoft SQL Server Null Values

Microsoft SQL Server has a special value called NULL. A Microsoft SQL Server NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Microsoft SQL Server NULL value, it interprets it as a SAS missing value.

Microsoft SQL Server columns can be defined as NOT NULL so that they require data—they cannot contain NULL values. When a column is defined as NOT NULL, the DBMS does not add a row to the table unless the row has a value for that column. When creating a DBMS table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how SAS missing character values are handled, use the NULLCHAR= and NULLCHARVAL= data set options.

LIBNAME Statement Data Conversions

This table shows all data types that SAS/ACCESS Interface to Microsoft SQL Server supports. This table also shows the default SAS format that corresponds to each data type.
Table 23.4  Microsoft SQL Server Data Types and Default SAS Formats

<table>
<thead>
<tr>
<th>Microsoft SQL Server Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_CHAR</td>
<td>$w$</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
<td>$w$</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td>$w$</td>
</tr>
<tr>
<td>SQL_BINARY</td>
<td>$w^*$</td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
<td>$w^*$</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td>$w^*$</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>$w$ or $w.d$ or none if $w$ and $d$ are not specified</td>
</tr>
<tr>
<td>SQL_NUMERIC</td>
<td>$w$ or $w.d$ or none if $w$ and $d$ are not specified</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>11.</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
<td>6.</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td>4.</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>1.</td>
</tr>
<tr>
<td>SQL_REAL</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FLOAT</td>
<td>none</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>none</td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td>20.</td>
</tr>
<tr>
<td>SQL_DATE</td>
<td>DATE9.</td>
</tr>
<tr>
<td>SQL_TIME</td>
<td>TIME8.</td>
</tr>
</tbody>
</table>

Microsoft SQL Server cannot support fractions of seconds for time values

SQL_TIMESTAMP                  | DATETIME$w,d$ where $w$ and $d$ depend on precision |

* Because the Microsoft SQL Server driver does the conversion, this field is displayed as if the $HEXw$ format were applied.

This table shows the default data types that the Microsoft SQL Server interface uses when creating tables.
### Table 23.5  Default Microsoft SQL Server Output Data Types

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Default Microsoft SQL Server Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.d$</td>
<td>SQL_DOUBLE or SQL_NUMERIC using $m,n$ if the DBMS allows it**</td>
</tr>
<tr>
<td>$w.$</td>
<td>SQL_VARCHAR using $n^*$</td>
</tr>
<tr>
<td>datetime formats</td>
<td>SQL_TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>SQL_DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>SQL_TIME</td>
</tr>
</tbody>
</table>

* $n$ in Microsoft SQL Server character data types is equivalent to $w$ in SAS formats.
** $m$ and $n$ in Microsoft SQL Server numeric data types are equivalent to $w$ and $d$ in SAS formats.

The Microsoft SQL Server interface allows non-default data types to be specified with the `DBTYPE=` on page 351 data set option.
### Introduction to SAS/ACCESS Interface to MySQL

For available SAS/ACCESS features, see MySQL supported features on page 83. For more information about MySQL, see your MySQL documentation.

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to MySQL</td>
<td>697</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for MySQL</td>
<td>698</td>
</tr>
<tr>
<td>Overview</td>
<td>698</td>
</tr>
<tr>
<td>Arguments</td>
<td>698</td>
</tr>
<tr>
<td>MySQL LIBNAME Statement Example</td>
<td>700</td>
</tr>
<tr>
<td>Data Set Options for MySQL</td>
<td>700</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for MySQL</td>
<td>702</td>
</tr>
<tr>
<td>Key Information</td>
<td>702</td>
</tr>
<tr>
<td>Examples</td>
<td>702</td>
</tr>
<tr>
<td>Autocommit and Table Types</td>
<td>703</td>
</tr>
<tr>
<td>Understanding MySQL Update and Delete Rules</td>
<td>703</td>
</tr>
<tr>
<td>Passing SAS Functions to MySQL</td>
<td>704</td>
</tr>
<tr>
<td>Passing Joins to MySQL</td>
<td>705</td>
</tr>
<tr>
<td>Bulk Loading for MySQL</td>
<td>705</td>
</tr>
<tr>
<td>Overview</td>
<td>705</td>
</tr>
<tr>
<td>Data Set Options for Bulk Loading</td>
<td>705</td>
</tr>
<tr>
<td>Examples</td>
<td>706</td>
</tr>
<tr>
<td>Naming Conventions for MySQL</td>
<td>706</td>
</tr>
<tr>
<td>Case Sensitivity for MySQL</td>
<td>707</td>
</tr>
<tr>
<td>Data Types for MySQL</td>
<td>707</td>
</tr>
<tr>
<td>Overview</td>
<td>707</td>
</tr>
<tr>
<td>Supported MySQL Data Types</td>
<td>707</td>
</tr>
<tr>
<td>LIBNAME Statement Data Conversions</td>
<td>708</td>
</tr>
</tbody>
</table>
LIBNAME Statement Specifics for MySQL

Overview

This section describes the LIBNAME statements that SAS/ACCESS Interface to MySQL supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing MySQL.

```
LIBNAME libref mysql <connection-options> <LIBNAME-options>;
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>libref</strong></td>
<td>specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables.</td>
</tr>
<tr>
<td><strong>mysql</strong></td>
<td>specifies the SAS/ACCESS engine name for MySQL interface.</td>
</tr>
<tr>
<td><strong>connection-options</strong></td>
<td>provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. Here is how these options are defined.</td>
</tr>
<tr>
<td>USER=</td>
<td>specifies the MySQL user login ID. If this argument is not specified, the current user is assumed. If the user name contains spaces or nonalphanumeric characters, you must enclose the user name in quotation marks.</td>
</tr>
<tr>
<td>PASSWORD=</td>
<td>specifies the MySQL password that is associated with the MySQL login ID. If the password contains spaces or nonalphanumeric characters, you must enclose the password in quotation marks.</td>
</tr>
<tr>
<td>DATABASE=</td>
<td>specifies the MySQL database to which you want to connect. If the database name contains spaces or nonalphanumeric characters, you must enclose the database name in quotation marks.</td>
</tr>
<tr>
<td>SERVER=</td>
<td>specifies the server name or IP address of the MySQL server. If the server name contains spaces or nonalphanumeric characters, you must enclose the server name in quotation marks.</td>
</tr>
<tr>
<td>PORT=port</td>
<td>specifies the port used to connect to the specified MySQL server.</td>
</tr>
<tr>
<td></td>
<td>Default: 3306</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>LIBNAME-options</strong></th>
<th>define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to MySQL, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.</th>
</tr>
</thead>
</table>
Table 24.1  SAS/ACCESS LIBNAME Options for MySQL

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_DEFAULT_DIR=</td>
<td>&lt;current-directory&gt;</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating, deleting, or appending rows to an existing table)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>ESCAPE_BACKSLASH=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>0</td>
</tr>
</tbody>
</table>
**MySQL LIBNAME Statement Example**

In the following example, the libref MYSQLLIB uses SAS/ACCESS Interface to MySQL to connect to a MySQL database. The SAS/ACCESS connection options are USER=, PASSWORD=, DATABASE=, SERVER=, and PORT=.

```sas
libname mysql libref MYSQLLIB user=myusr1 password=mypwd1 database=mysqldb server=mysrv1 port=9876;

proc print data=mysql.employees;
  where dept='CSR010';
run;
```

## Data Set Options for MySQL

All SAS/ACCESS data set options in this table are supported for MySQL. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

### Table 24.2  Data Set Options for MySQL

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOCOMMIT=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>BL_DEFAULT_DIR=</td>
<td>&lt;current-directory&gt;</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for MySQL”</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “LIBNAME Statement Data Conversions”</td>
</tr>
<tr>
<td>ESCAPE_BACKSLASH=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>0</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>current LIBNAME option setting</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>DATETIME20.0</td>
</tr>
<tr>
<td>UPDATE_ISOATION_LEVEL=</td>
<td>the current LIBNAME option setting</td>
</tr>
</tbody>
</table>
SQL Pass-Through Facility Specifics for MySQL

Key Information

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for MySQL.

- The dbms-name is mysql.
- If you call MySQL stored procedures that return multiple result sets, SAS returns only the last result set.
- The CONNECT statement database-connection-arguments are identical to its LIBNAME connection options.

Note: Due to a current limitation in the MySQL client library, you cannot run MySQL stored procedures when SAS is running on AIX.

Examples

This example uses the alias DBCON for the DBMS connection (the connection alias is optional):

```sas
proc sql;
  connect to mysql as dbcon
    (user=myusr1 password=mypwd1 server=mysrv1
database=mysqldb port=9876);
quit;
```

This example connects to MySQL and sends it two EXECUTE statements to process:

```sas
proc sql;
  connect to mysql (user=myusr1 password=mypwd1 server=mysrv1
database=mysqldb port=9876);
  execute (create table whotookorders as
    select ordernum, takenby,
    firstname, lastname, phone
    from orders, employees
    where orders.takenby=employees.empid)
  by mysql;
  execute (grant select on whotookorders
    to myusr1) by mysql;
  disconnect from mysql;
quit;
```

This example performs a query, shown in highlighted text, on the MySQL table CUSTOMERS:

```sas
proc sql;
  connect to mysql (user=myusr1 password=mypwd1 server=mysrv1
database=mysqldb port=9876);
  select *
    from connection to mysql
  (select * from customers
   where...)}
```
where customer like '1%');

disconnect from mysql;
quit;

Autocommit and Table Types

MySQL supports several table types, two of which are InnoDB (the default) and MyISAM. A single database can contain tables of different types. The behavior of a table is determined by its table type. For example, by definition, a table created of MyISAM type does not support transactions. Consequently, all DML statements (updates, deletes, inserts) are automatically committed. If you need transactional support, specify a table type of InnoDB in the DBCREATE_TABLE_OPTS=LIBNAME option. This table type allows for updates, deletes, and inserts to be rolled back if an error occurs; or updates, deletes, and inserts to be committed if the SAS DATA step or procedure completes successfully.

By default, the MYSQL LIBNAME engine sets AUTOCOMMIT=YES regardless of the table type. If you are using tables of the type InnoDB, set AUTOCOMMIT=NO to improve performance. To control how often COMMITS are executed, set the DBCOMMIT=option.

Note: The DBCOMMIT option can affect SAS/ACCESS performance. Experiment with a value that best fits your table size and performance needs before using it for production jobs. Transactional tables require significantly more memory and disk space requirements.

Understanding MySQL Update and Delete Rules

To avoid data integrity problems when updating or deleting data, you need to define a primary key on your table. See MySQL documentation for more information about table types and transactions.

This example uses AUTOCOMMIT=NO and DBTYPE= to create the primary key and also DBCREATE_TABLE_OPTS= to determine the MySQL table type.

libname invty mysql user=myusr1 server=mysrv1 database=test autocommit=no reread_exposure=no;

proc sql;
drop table invty.STOCK23;
quit;

/* Create DBMS table with primary key and of type INNODB */
data invty.STOCK23(drop=PARTNO DBTYPE=(RECDATE="date not null,
primary key(RECDATE)") DBCREATE_TABLE_OPTS="type = innodb");
input PARTNO $ DESCX $ INSTOCK @17
RECDATE date7. @25 PRICE;
format RECDATE date7.;
datalines;
K89R  seal     34  27jul95  245.00
M447  sander   98  20jun95   45.88
LK43  filter  121  19may96   10.99
This next example shows how you can update the table now that STOCK23 has a primary key.

```sas
proc sql;
update invty.STOCK23 set price=price*1.1 where INSTOCK > 50;
quit;
```

### Passing SAS Functions to MySQL

`SQL_FUNCTIONS=ALL` allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when `SQL_FUNCTIONS=ALL` can the SAS/ACCESS engine also pass these SAS SQL functions to MySQL. Due to incompatibility in date and time functions between MySQL and SAS, MySQL might not process them correctly. Check your results to determine whether these functions are working as expected.

Where the MySQL function name differs from the SAS function name, the MySQL name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

By default, the following SAS functions are passed down to the database:

- `ABS`
- `ARCOS (ACOS)`
- `ARCSIN (ASIN)`
- `ATAN`
- `ATAN2`
- `AVG`
- `CEIL (CEILING)`
- `COALESCE`
- `COS`
- `COT`
- `COUNT`
- `DAY (DAYOFMONTH)`
- `EXP`
- `FLOOR`
- `HOUR`
- `INDEX (LOCATE)`
- `LOG`
- `LOG2`
- `LOG10`
- `LOG2 (LOG)`
- `LOG10 (LOG)`
- `LOWCASE (LCASE)`
- `MAX`
- `MIN`
- `MINUTE`
- `MOD (see note)`
- `MONTH`
- `QTR (QUARTER)`
- `SECOND`
- `SIGN`
- `SIN`
- `SQRT`
- `STRIP (TRIM)`
- `TRANWRD (REPLACE)`
- `TRIMN (RTRIM)`
- `UPCASE (UCASE)`
- `WEEKDAY (DAYOFWEEK)`
- `YEAR`
Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see "Functions Where Results Might Vary: MOD Function" on page 44.

The following SAS functions are passed down to the database only if SQL_FUNCTIONS=ALL:

- BYTE (CHAR)
- COMPRESS (REPLACE)
- DATE (CURDATE)
- DATEPART
- DATETIME (NOW)
- LENGTH
- REPEAT
- ROUND
- SOUNDEX
- SUBSTR (SUBSTRING)
- TIME (CURTIME())
- TIMEPART
- TODAY (CURDATE())

Passing Joins to MySQL

For a multiple libref join to pass to MySQL, all of these components of the LIBNAME statements must match exactly:

- user (USER=)
- password (PASSWORD=)
- database DATABASE=)
- server (SERVER=)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for MySQL

Overview

Bulk loading is the fastest way to insert large numbers of rows into a MySQL table. Using this facility instead of regular SQL insert statements, you can insert rows more rapidly.

The MySQL LIBNAME engine calls the MySQL bulk-load facility when you set BULKLOAD=YES. You must specify BULKLOAD=YES to use this facility.

This facility uses the LOAD DATA command to create a text file on the client, which lets MySQL bulk load it into a table on the server.

Data Set Options for Bulk Loading

Here are the MySQL bulk-load data set options. For detailed information about these options, see Data Set Options for Relational Databases on page 241.

- BL_DEFAULT_DIR=
Examples

This example uses the DATA step.

```sas
libname x mysql user=myusr1 server=mysrv1 database=mydb1 password=mypwd1;
run;

data x.mydata_bulk21(BL_DEFAULT_DIR="c:/temp/mysql" bulkload=yes);
col1=1;col2='Test';
run;
```

This next example uses PROC APPEND.

```sas
libname x mysql user=myusr1 server=mysrv1 database=mydb1 password=mypwd1;
run;

data x;col1=1;col2='TEST';
output;
run;

proc append data=x base=x.mydata_bulk21(bulkload=yes);
run;
```

Naming Conventions for MySQL

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

MySQL database identifiers that you can name include databases, tables, and columns. They follow these naming conventions.

- Aliases must be from 1 to 255 characters long. All other identifier names must be from 1 to 64 characters long.
- Database names can use any character that is allowed in a directory name except for a period, a backward slash (\), or a forward slash (/).
- By default, MySQL encloses column names and table names in quotation marks.
- Table names can use any character that is allowed in a filename except for a period or a forward slash.
- Table names must be 32 characters or less because SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.
- Column names and alias names allow all characters.
- Embedded spaces and other special characters are not permitted unless you enclose the name in quotation marks.
- Embedded quotation marks are not permitted.
- Case sensitivity is set when a server is installed. By default, the names of database objects are case sensitive on UNIX and not case sensitive on Windows. For example, the names `CUSTOMER` and `Customer` are different on a case-sensitive server.
• A name cannot be a reserved word in MySQL unless you enclose the name in quotation marks. See the MySQL documentation for more information about reserved words.

• Database names must be unique. For each user within a database, names of database objects must be unique across all users. For example, if a database contains a department table that User A created, no other user can create a department table in the same database.

MySQL does not recognize the notion of schema, so tables are automatically visible to all users with the appropriate privileges. Column names and index names must be unique within a table.

---

**Case Sensitivity for MySQL**

In MySQL, databases and tables correspond to directories and files within those directories. Consequently, the case sensitivity of the underlying operating system determines the case sensitivity of database and table names. This means database and table names are not case sensitive in Windows, and case sensitive in most varieties of UNIX.

In SAS, names can be entered in either uppercase or lowercase. MySQL recommends that you adopt a consistent convention of either all uppercase or all lowercase table names, especially on UNIX hosts. This can be easily implemented by starting your server with `-O lower_case_table_names=1`. Please see the MySQL documentation for more details.

If your server is on a case-sensitive platform and you choose to allow case sensitivity, be aware that when you reference MYSQL objects through the SAS/ACCESS interface, objects are case sensitive and require no quotation marks. Also, in the SQL pass-through facility, all MySQL object names are case sensitive. Names are passed to MySQL exactly as they are entered.

For more information about case sensitivity and MySQL names, see “Naming Conventions for MySQL” on page 706.

---

**Data Types for MySQL**

**Overview**

Every column in a table has a name and a data type. The data type tells MySQL how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about MySQL data types and data conversions.

**Supported MySQL Data Types**

Here are the data types that the MySQL engine supports.

• Character data:

  BLOB (binary large object)  MEDIUMTEXT
CHAR \( (n) \) 

SET ("value1", "value2", "value3", …)

ENUM ("value1", "value2", "value3", …)

TEXT

LONGBLOB 

TINYBLOB 

LONGTEXT 

TINYTEXT 

MEDIUMBLOB 

VARCHAR \( (n) \)

• Numeric data:

BIGINT \( (n) \) 

INT \( (n) \)

DECIMAL (length, decimals) 

MEDIUMINT \( (n) \)

DOUBLE (length, decimals) 

SMALLINT \( (n) \)

FLOAT (length, decimals) 

TINYINT \( (n) \)

Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

• Date, time, and timestamp data:

DATE 

TIME 

DATETIME 

TIMESTAMP

LIBNAME Statement Data Conversions

This table shows the default formats that SAS/ACCESS Interface to MySQL assigns to SAS variables when using the LIBNAME statement to read from a MySQL table. These default formats are based on MySQL column attributes.

Table 24.3 LIBNAME Statement: Default SAS Formats for MySQL Data Types

<table>
<thead>
<tr>
<th>MySQL Column Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR((n))*</td>
<td>character</td>
<td>(w)</td>
</tr>
<tr>
<td>VARCHAR((n))*</td>
<td>character</td>
<td>(w)</td>
</tr>
<tr>
<td>TINYTEXT</td>
<td>character</td>
<td>(w)</td>
</tr>
<tr>
<td>TEXT</td>
<td>character</td>
<td>(w.)'</td>
</tr>
<tr>
<td>MEDIUMMTEXT</td>
<td>character</td>
<td>(w.)''</td>
</tr>
<tr>
<td>LONGTEXT</td>
<td>character</td>
<td>(w.)''</td>
</tr>
<tr>
<td>TINYBLOB</td>
<td>character</td>
<td>(w.)''</td>
</tr>
</tbody>
</table>
This table shows the default MySQL data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

<table>
<thead>
<tr>
<th>MySQL Column Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>character</td>
<td>$w.$</td>
</tr>
<tr>
<td>MEDIUMBLOB</td>
<td>character</td>
<td>$w.$</td>
</tr>
<tr>
<td>LONGBLOB</td>
<td>character</td>
<td>$w.$</td>
</tr>
<tr>
<td>ENUM</td>
<td>character</td>
<td>$w.$</td>
</tr>
<tr>
<td>SET</td>
<td>character</td>
<td>$w.$</td>
</tr>
<tr>
<td>TINYINT</td>
<td>numeric</td>
<td>4.0</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.0</td>
</tr>
<tr>
<td>MEDIUMINT</td>
<td>numeric</td>
<td>8.0</td>
</tr>
<tr>
<td>INT</td>
<td>numeric</td>
<td>11.0</td>
</tr>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>numeric</td>
<td>$w.d$</td>
</tr>
<tr>
<td>FLOAT</td>
<td>numeric</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>numeric</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME</td>
</tr>
<tr>
<td>DATETIME</td>
<td>numeric</td>
<td>DATETIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME</td>
</tr>
</tbody>
</table>

* $n$ in MySQL character data types is equivalent to $w$ in SAS formats.

** In this case, $w$ is the value of the DBMAX_TEXT= option.
<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>MySQL Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$ (where $w \leq 6$)</td>
<td>MEDIUMINT</td>
</tr>
<tr>
<td>$w$ (where $w \leq 17$)</td>
<td>BIGINT</td>
</tr>
<tr>
<td>other numerics</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>$sw$ (where $w \leq 255$)</td>
<td>VARCHAR($n$)*</td>
</tr>
<tr>
<td>$sw$ (where $w &gt; 255$)</td>
<td>TEXT</td>
</tr>
<tr>
<td>datetime formats</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>TIME</td>
</tr>
</tbody>
</table>

* $n$ in MySQL character data types is equivalent to $w$ in SAS formats.
** $m$ and $n$ in MySQL numeric data types are equivalent to $w$ and $d$ in SAS formats.
*** DECIMAL types are created as $(m-1, n)$. SAS includes space to write the value, the decimal point, and a minus sign (if necessary) in its calculation for precision. These must be removed when converting to MySQL.
Introduction to SAS/ACCESS Interface to Netezza

For available SAS/ACCESS features, see Netezza supported features on page 84. For more information about Netezza, see your Netezza documentation.
LIBNAME Statement Specifics for Netezza

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Netezza supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Netezza.

```
LIBNAME libref netezza <connection-options> <LIBNAME-options>;
```

Arguments

`libref`

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

`netezza`

specifies the SAS/ACCESS engine name for the Netezza interface.

`connection-options`

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to the Netezza Performance Server in one of two ways. Specify only one of these methods for each connection because they are mutually exclusive.

- `SERVER=, DATABASE=, PORT=, USER=, PASSWORD=, READ_ONLY=`
- `DSN=, USER=, PASSWORD=`

Here is how these options are defined.

*Note:* All of the following connection options are also valid in the CONNECT statement when you use the SQL pass-through facility (SQL procedure) to connect to your DBMS.

`SERVER=<server-name>`

specifies the server name or IP address of the Netezza Performance Server to which you want to connect. This server accesses the database that contains the tables and views that you want to access. If the server name contains spaces or nonalphanumeric characters or if it is an IP address, you must enclose it in quotation marks.

`DATABASE=<database-name>`

specifies the name of the database on the Netezza Performance Server that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: `DB=`

`PORT=port`

specifies the port number that is used to connect to the specified Netezza Performance Server. If you do not specify a port, the default is 5480.
specifies the Netezza user name (also called the user ID) that you use to connect
to your database. If the user name contains spaces or nonalphanumeric
characters, you must enclose it in quotation marks.

specifies the password that is associated with your Netezza user name. If the
password contains spaces or nonalphanumeric characters, you must enclose it in
quotation marks.

Alias: PASS=, PW=, PWD=

specifies whether to connect to the Netezza database in Read-Only mode (YES)
or read-write (NO) mode. If you do not specify anything for READ_ONLY=, the
default of NO is used.

Alias: READONLY=

specifies the configured Netezza ODBC data source to which you want to
connect. Use this option if you have existing Netezza ODBC data sources that
are configured on your client. This method requires additional setup—either
through the ODBC Administrator control panel on Windows platforms or
through the odbc.ini file or a similarly named configuration file on UNIX
platforms. It is recommended that you use this connection method only if you
have existing, functioning data sources that have been defined.

LIBNAME-options
define how SAS processes DBMS objects. Some LIBNAME options can enhance
performance, and others determine locking or naming behavior. The following table
describes the LIBNAME options for SAS/ACCESS Interface to Netezza, with the
applicable default values. This table also identifies LIBNAME options that are valid
in the CONNECT statement in the SQL procedure. For more information, see
LIBNAME Options for Relational Databases on page 101 or Chapter 14, “SQL Pass-
Through Facility for Relational Databases,” on page 471.

Table 25.1  SAS/ACCESS LIBNAME Options for Netezza

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
<th>Valid in CONNECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>operation-specific</td>
<td>●</td>
</tr>
<tr>
<td>BULKUNLOAD=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>CHAR_AS_NCHAR=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>UNIQUE</td>
<td>●</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 when inserting rows, 0 when updating rows</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
<td>Valid in CONNECT</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>------------------</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
<td>●</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DLBLIBINIT=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DLBLIBTERM=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
<td>●</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
<td></td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>automatically calculated based on row length</td>
<td></td>
</tr>
<tr>
<td>LOGIN_TIMEOUT=</td>
<td>0</td>
<td>●</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for Netezza”</td>
<td></td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>see “Naming Conventions for Netezza”</td>
<td></td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
### Option Table

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
<th>Valid in CONNECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
<td>●</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>READBUFF=</td>
<td>automatically calculated based on row length</td>
<td>●</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>the current connection schema</td>
<td></td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>SYNONYMS=</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>USE_ODBC_CL=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

### Netezza LIBNAME Statement Examples

In this example, SERVER=, DATABASE=, USER=, and PASSWORD= are connection options.

```plaintext
libname mydblib netezza server=mysrv11 database=test
  user=myuser password=mypwd;

proc print data=mydblib.customers;
  where state='CA';
run;
```

In the next example, DSN=, USER=, and PASSWORD= are connection options. The NZSQL data source is configured in the ODBC Administrator Control Panel on Windows platforms or in the odbc.ini file or a similarly named configuration file on UNIX platforms.

```plaintext
libname mydblib netezza dsn=NZSQL
  user=myuser password=mypwd;
```
proc print data=mydblib.customers;
  where state='CA';
run;

Data Set Options for Netezza

SAS/ACCESS data set options in this table are supported for Netezza. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

Table 25.2 SAS/ACCESS Data Set Options for Netezza

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_DATAFILE=</td>
<td>When BL_USE_PIPE=NO, creates a file in the current directory or with the default file specifications.</td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE=</td>
<td>YES (only when BL_USE_PIPE=NO)</td>
</tr>
<tr>
<td>BL_DELIMITER=</td>
<td></td>
</tr>
<tr>
<td>BL_OPTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>BL_USE_PIPE=</td>
<td>YES</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>BULKUNLOAD=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for Netezza”</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for Netezza”</td>
</tr>
<tr>
<td>DISTRIBUT_ON=</td>
<td>none</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>
SQL Pass-Through Facility Specifics for Netezza

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Netezza interface.

- The `dbms-name` is **NETEZZA**.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to Netezza. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default `netezza` alias is used.
- The CONNECT statement `database-connection-arguments` are identical to its LIBNAME `connection-options`. Additional LIBNAME options that are valid in the CONNECT statement are indicated in Table 25.1 on page 713.

**CONNECT Statement Example**

This example uses the DBCON alias to connection to the `mysrv1` Netezza Performance Server and execute a query. The connection alias is optional.

```sql
proc sql;
  connect to netezza as dbcon
    (server=mysrv1 database=test user=myuser password=mypwd);
  select * from connection to dbcon
    (select * from customers where customer like '1%');
quit;
```

**Special Catalog Queries**

SAS/ACCESS Interface to Netezza supports the following special queries. You can the queries use to call the ODBC-style catalog function application programming interfaces (APIs). Here is the general format of the special queries:

Netezza::SQLAPI "parameter 1","parameter n"

Netezza:: is required to distinguish special queries from regular queries.

SQLAPI is the specific API that is being called. Neither Netezza:: nor SQLAPI are case sensitive.

"parameter n" is a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters, and the underscore represents any single character. To use either character as a literal value, you can use the backslash character (\) to escape the match characters. For
example, this call to SQLTables usually matches table names such as mytest and my_test:

```
select * from connection to netezza (NETEZZA::SQLTables "test",**,"my_test");
```

Use the escape character to search only for the my_test table:

```
select * from connection to netezza (NETEZZA::SQLTables "test",**,"my\_test");
```

SAS/ACCESS Interface to Netezza supports these special queries:

Netezza::SQLTables <"Catalog", "Schema", "Table-name", "Type"> returns a list of all tables that match the specified arguments. If you do not specify any arguments, all accessible table names and information are returned.

Netezza::SQLColumns <"Catalog", "Schema", "Table-name", "Column-name"> returns a list of all columns that match the specified arguments. If you do not specify any argument, all accessible column names and information are returned.

Netezza::SQLPrimaryKeys <"Catalog", "Schema", "Table-name"> returns a list of all columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If you do not specify any table name, this special query fails.

Netezza::SQLSpecialColumns <"Identifier-type", "Catalog-name", "Schema-name", "Table-name", "Scope", "Nullable"> returns a list of the optimal set of columns that uniquely identify a row in the specified table.

Netezza::SQLStatistics <"Catalog", "Schema", "Table-name"> returns a list of the statistics for the specified table name. You can set the SQL_INDEX_ALL and SQL_ENSURE options in the SQLStatistics API call. If you do not specify any table name argument, this special query fails.

Netezza::SQLGetTypeInfo returns information about the data types that the Netezza Performance Server supports.

---

**Temporary Table Support for Netezza**

SAS/ACCESS Interface to Netezza supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

---

**Passing SAS Functions to Netezza**

SAS/ACCESS Interface to Netezza passes the following SAS functions to Netezza for processing. Where the Netezza function name differs from the SAS function name, the Netezza name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

- `ABS` to `LOG10 (log)`
- `ARCOS (ACOS)` to `LOWCASE (lower)`
- `ARSIN (ASIN)` to `MAX`
- `ATAN` to `MIN`
ATAN2 MINUTE (date_part)
AVG MOD (see note)
BAND (int4and) MONTH (date_part)
BNOT (int4not) QTR (date_part)
BLSHIFT (int4shl) REPEAT
BRSHIFT (int4shr) SECOND (date_part)
BOR (int4or) SIGN
BXOR (int4xor) SIN
BYTE (chr) SQRT
CEIL STRIP (btrim)
COALESCE SUBSTR
COMPRESS (translate) SUM
COS TAN
COUNT TRANWRD (translate)
DAY (date_part) TRIMN (rtrim)
EXP UPCASE (upper)
FLOOR WEEK[<SAS date value>, V'] (date_part)
HOUR WEEKDAY (date_part)
INDEX (position) YEAR (date_part)
LOG (ln)

Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Netezza. Due to incompatibility in date and time functions between Netezza and SAS, Netezza might not process them correctly. Check your results to determine whether these functions are working as expected.

DATE (current_date) TODAY (current_date)
DATEPART (cast) TRANSLATE
DATETIME (now) WEEK[<SAS date value>] (date_part)
LENGTH WEEK[<SAS date value>, 'U'] (date_part)
ROUND WEEK[<SAS date value>, 'W'] (date_part)
TIME (current_time) WEEKDAY (date_part)
TIMEPART (cast)

Passing Joins to Netezza

For a multiple libref join to pass to Netezza, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
Bulk Loading and Unloading for Netezza

Loading

Overview
Bulk loading is the fastest way to insert large numbers of rows into a Netezza table. You must specify BULKLOAD=YES to use the bulk-load facility. The bulk-load facility uses the Netezza Remote External Table interface to move data from the client to the Netezza Performance Server.

Data Set Options for Bulk Loading
Here are the Netezza bulk-load data set options. For detailed information about these options, see Data Set Options for Relational Databases on page 241.

- BL_DATAFILE=
- BL_DELETE_DATAFILE=
- BL_DELIMITER=
- BL_OPTIONS=
- BL_USE_PIPE=
- BULKLOAD=

Examples
This first example shows how you can use a SAS data set, SASFLT.FLT98, to create and load a large Netezza table, FLIGHTS98:

```
libname sasflt 'SAS-library';
libname net_air netezza user=myuser pwd=mypwd
  server=air2 database=flights;
proc sql;
create table net_air.flights98
  (bulkload=YES bl_options="logdir 'c:\temp\netlogs'")
  as select * from sasflt.flt98;
quit;
```

You can use BL_OPTIONS= to pass specific Netezza options to the bulk-loading process. The LOGDIR option specifies the directory for the Nzbad and Nzlog files to be generated during the load.

This next example shows how you can append the SAS data set, SASFLT.FLT98, to the existing Netezza table, ALLFLIGHTS. The BL_USE_PIPE=NO option forces
SAS/ACCESS Interface to Netezza to write data to a flat file, as specified in the BL_DATAFILE= option. Rather than deleting the data file, BL_DELETE_DATAFILE=NO causes the engine to leave it after the load has completed.

```
proc append base=net_air.allflights
   (BULKLOAD=YES
    BL_DATAFILE='/tmp/fltdata.dat'
    BL_USE_PIPE=NO
    BL_DELETE_DATAFILE=NO)
data=sasflt.flt98;
run;
```

**Unloading**

**Overview**

Bulk unloading is the fastest way to insert large numbers of rows from a Netezza table. To use the bulk-unloading facility, specify BULKUNLOAD=YES. (See BULKUNLOAD= on page 318.) The bulk-unloading facility uses the Netezza Remote External Table interface to move data from the client to the Netezza Performance Server into SAS.

**Data Set Options for Bulk Unloading**

Here are the Netezza bulk-unloading data set options.

- **BL_DATAFILE=**
- **BL_DELETE_DATAFILE=**
- **BL_DELIMITER=**
- **BL_USE_PIPE=**
- **BULKLOAD=**

**Examples**

This first example shows how you can read the large Netezza table, FLIGHTS98, to create and populate a SAS data set, SASFLT.FLT98:

```
libname sasflt 'SAS-library';
libname net_air netezza user=myuser pwd=mypwd
    server=air2 database=flights;
proc sql;
create table sasflt.flt98
   as select * from net_air.flights98
   (bulkunload=YES bl_options="logdir 'c:\temp\netlogs'"));
quit;
```

You can use BL_OPTIONS= to pass specific Netezza options to the bulk unloading process. The LOGDIR option specifies the directory for the Nzbad and Nzlog files to be generated during the bulk unloading.

This next example shows how you can append the contents of the Netezza table, ALLFLIGHTS, to an existing SAS data set, SASFLT.FLT98. The BL_USE_PIPE=NO option forces SAS/ACCESS Interface to Netezza to read data from a flat file, as specified in the BL_DATAFILE= option. Rather than deleting the data file,
BL_DELETE_DATAFILE=NO causes the engine to leave it after the bulk unloading has completed.

```sas
proc append base=sasflt.flt98
    data=net_air.allflights
    (BULKUNLOAD=YES
      BL_DATAFILE='/tmp/fltdata.dat'
      BL_USE_PIPE=NO
      BL_DELETE_DATAFILE=NO);
run;
```

### Naming Conventions for Netezza

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The Netezza interface supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name would result in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

The PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options determine how SAS/ACCESS Interface to Netezza handles case sensitivity. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Netezza is not case sensitive, and all names default to lowercase.

Netezza objects include tables, views, and columns. Follow these naming conventions:

- A name must be from 1 to 128 characters long.
- A name must begin with a letter (A through Z), diacritic marks, or non-Latin characters (200-377 octal).
- A name cannot begin with an underscore (_). Leading underscores are reserved for system objects.
- Names are not case sensitive. For example, `CUSTOMER` and `Customer` are the same, but object names are converted to lowercase when they are stored in the Netezza database. However, if you enclose a name in quotation marks, it is case sensitive.
- A name cannot be a Netezza reserved word, such as `WHERE` or `VIEW`.
- A name cannot be the same as another Netezza object that has the same type.

For more information, see your *Netezza Database User's Guide*.

### Data Types for Netezza

#### Overview

Every column in a table has a name and a data type. The data type tells Netezza how much physical storage to set aside for the column and the form in which the data is
stored. This section includes information about Netezza data types, null and default values, and data conversions.

For more information about Netezza data types and to determine which data types are available for your version of Netezza, see your Netezza Database User’s Guide.

SAS/ACCESS Interface to Netezza does not directly support TIMETZ or INTERVAL types. Any columns using these types are read into SAS as character strings.

**Supported Netezza Data Types**

Here are the data types that are supported by Netezza:

- **Character data:**
  - CHAR(n)
  - NVARCHAR(n)
  - NCHAR(n)
  - VARCHAR(n)

- **Numeric data:**
  - BIGINT
  - REAL
  - BYTEINT
  - SMALLINT
  - DECIMAL | DEC | NUMERIC | NUM
  - ST_GEOMETRY
  - DOUBLE | DOUBLE PRECISION
  - VARBINARY
  - INTEGER

*Note:* Support for ST_GEOMETRY and VARBINARY was added in the second maintenance release for SAS 9.4.

You might observe errors in the SAS log when you load large numeric values with more than 15 digits of precision. For more information, see “Loading Large Numeric Values” on page 724.

Columns that use the binary data types, such as ST_GEOMETRY and VARBINARY, do not support some of the common query processing operations. For example, binary data type columns cannot be used in ordering, grouping, or in magnitude comparisons. They cannot be used in aggregates such as sum, avg, distinct, min, or max comparisons. The binary data cannot be implicitly or explicitly cast to other data types.

- **Date and time data:**
  - DATE
  - TIME
  - DATETIME

SQL date and time data types are collectively called datetime values. The SQL data types for dates, times, and timestamps are listed here. Be aware that columns of these data types can contain data values that are out of range for SAS.

**Loading Large Numeric Values**

When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you
can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

By default, when SAS/ACCESS loads numeric data with more than 15 digits of precision, errors are written to the log and the data fails to load. You can choose to load the data and suppress the errors by setting the TRUNCATE_BIGINT environment variable to YES before you start SAS.

**Netezza Null Values**

Netezza has a special value called NULL. A Netezza NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Netezza NULL value, it interprets it as a SAS missing value.

You can define a column in a Netezza table so that it requires data. To do this in SQL, you specify a column as NOT NULL. This tells SQL to allow only a row to be added to a table if a value exists for the field. For example, NOT NULL assigned to the CUSTOMER field in the SASDEMO.CUSTOMER table does not allow a row to be added unless there is a value for CUSTOMER. When creating a Netezza table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

You can also define Netezza columns as NOT NULL DEFAULT. For more information about using the NOT NULL DEFAULT value, see your *Netezza Database User's Guide*.

Once you know whether a Netezza column enables NULLs or the host system supplies a default value for a column that is defined as NOT NULL WITH DEFAULT, you can write selection criteria and enter values to update a table. Unless a column is defined as NOT NULL or NOT NULL DEFAULT, it allows NULL values.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how the DBMS handles SAS missing character values, use the NULLCHAR= and NULLCHARVAL= data set options.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to Netezza assigns to SAS variables when using the LIBNAME statement to read from a Netezza table. These default formats are based on Netezza column attributes.

<table>
<thead>
<tr>
<th>Netezza Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>CHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>DECIMAL(p,s)**</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>Netezza Data Type</td>
<td>SAS Data Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>INTEGER</td>
<td>numeric</td>
<td>11.</td>
</tr>
<tr>
<td>NCHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>NUMERIC(p,s)**</td>
<td>numeric</td>
<td>w.d</td>
</tr>
<tr>
<td>NVARCHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.</td>
</tr>
<tr>
<td>BYTESINT</td>
<td>numeric</td>
<td>4.</td>
</tr>
<tr>
<td>ST_GEOMETRY</td>
<td>character</td>
<td>$HEX2w.</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME8.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME25.6</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>character</td>
<td>$HEX2w.</td>
</tr>
<tr>
<td>VARCHAR(n)*</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>VINBINARY</td>
<td>numeric</td>
<td>8.</td>
</tr>
</tbody>
</table>

* $n$ in Netezza character data types is equivalent to $w$ in SAS formats.
** $p$ and $s$ in Netezza numeric data types are equivalent to $w$ and $d$ in SAS formats.

The following table shows the default Netezza data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

**Table 25.4 LIBNAME Statement: Default Netezza Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Netezza Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.d</td>
<td>DECIMAL(p,s)**</td>
</tr>
<tr>
<td>other numerics</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>$w.$</td>
<td>VARCHAR(n)*</td>
</tr>
<tr>
<td>datetime formats</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>TIME</td>
</tr>
</tbody>
</table>

* $n$ in Netezza character data types is equivalent to $w$ in SAS formats.
** $p$ and $s$ in Netezza numeric data types are equivalent to $w$ and $d$ in SAS formats.
# Chapter 26

SAS/ACCESS Interface to ODBC

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to SAS/ACCESS Interface to ODBC</td>
<td>728</td>
</tr>
<tr>
<td>Overview</td>
<td>728</td>
</tr>
<tr>
<td>ODBC Concepts</td>
<td>728</td>
</tr>
<tr>
<td>LIBNAME Statement Specifics for ODBC</td>
<td>732</td>
</tr>
<tr>
<td>Overview</td>
<td>732</td>
</tr>
<tr>
<td>Arguments</td>
<td>732</td>
</tr>
<tr>
<td>ODBC LIBNAME Statement Examples</td>
<td>736</td>
</tr>
<tr>
<td>Data Set Options for ODBC</td>
<td>737</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for ODBC</td>
<td>738</td>
</tr>
<tr>
<td>Key Information</td>
<td>738</td>
</tr>
<tr>
<td>CONNECT Statement Examples</td>
<td>739</td>
</tr>
<tr>
<td>Connection to Component Examples</td>
<td>740</td>
</tr>
<tr>
<td>Special Catalog Queries</td>
<td>741</td>
</tr>
<tr>
<td>Autopartitioning Scheme for ODBC</td>
<td>743</td>
</tr>
<tr>
<td>Overview</td>
<td>743</td>
</tr>
<tr>
<td>Autopartitioning Restrictions</td>
<td>743</td>
</tr>
<tr>
<td>Nullable Columns</td>
<td>743</td>
</tr>
<tr>
<td>Using WHERE Clauses</td>
<td>743</td>
</tr>
<tr>
<td>Using DBSLICEPARM=</td>
<td>744</td>
</tr>
<tr>
<td>Using DBSLICE=</td>
<td>744</td>
</tr>
<tr>
<td>Configuring SQL Server Partitioned Views for Use with DBSLICE=</td>
<td>744</td>
</tr>
<tr>
<td>DBLOAD Procedure Specifics for ODBC</td>
<td>747</td>
</tr>
<tr>
<td>Overview</td>
<td>747</td>
</tr>
<tr>
<td>Examples</td>
<td>748</td>
</tr>
<tr>
<td>Temporary Table Support for ODBC</td>
<td>749</td>
</tr>
<tr>
<td>Passing SAS Functions to ODBC</td>
<td>749</td>
</tr>
<tr>
<td>Passing Joins to ODBC</td>
<td>750</td>
</tr>
<tr>
<td>Bulk Loading for ODBC</td>
<td>750</td>
</tr>
<tr>
<td>Locking in the ODBC Interface</td>
<td>750</td>
</tr>
<tr>
<td>Naming Conventions for ODBC</td>
<td>752</td>
</tr>
<tr>
<td>Data Types for ODBC</td>
<td>752</td>
</tr>
<tr>
<td>Overview</td>
<td>752</td>
</tr>
<tr>
<td>ODBC Null Values</td>
<td>753</td>
</tr>
<tr>
<td>LIBNAME Statement Data Conversions</td>
<td>753</td>
</tr>
</tbody>
</table>
Introduction to SAS/ACCESS Interface to ODBC

Overview

For available SAS/ACCESS features, see ODBC supported features on page 85. For more information about ODBC, see your ODBC documentation.

ODBC Concepts

Overview

Open database connectivity (ODBC) standards provide a common interface to a variety of data sources, including dBASE, Microsoft Access, Microsoft SQL Server, Oracle, and Paradox. The goal of ODBC is to enable access to data from any application, regardless of which DBMS handles the data. ODBC accomplishes this by inserting a middle layer—consisting of an ODBC driver manager and an ODBC driver—between an application and the target DBMS. The purpose of this layer is to translate application data queries into commands that the DBMS understands. Specifically, ODBC standards define application programming interfaces (APIs) that enable applications such as SAS software to access a database. For all of this to work, both the application and the DBMS must be ODBC-compliant. This means that the application must be able to issue ODBC commands, and the DBMS must be able to respond to these.

Here are the basic components and features of ODBC.

Three components provide ODBC functionality: the client interface, the ODBC driver manager, and the ODBC driver for the data source with which you want to work, as shown below.

Figure 26.1 The ODBC Interface to SAS

For PC and UNIX environments, SAS provides SAS/ACCESS Interface to ODBC as the client interface. Consisting of the ODBC driver manager and the ODBC driver, the client setup with which SAS/ACCESS Interface to ODBC works is quite different between the two platforms.

ODBC on a PC Platform

On the PC side, the Microsoft ODBC Data Source Administrator is the ODBC driver manager. You can open the ODBC Data Source Administrator from the Windows control...
Through a series of dialog boxes, you can create an ODBC data source name (DSN). You can select a particular ODBC driver for the database with which you want to work from the list of available drivers. You can then provide specific connection information for the database that the specific driver can access.

**USER DSN**
- specific to an individual user. It is available only to the user who creates it.

**SYSTEM DSN**
- not specific to an individual user. Anyone with permission to access the data source can use it.

**FILE DSN**
- not specific to an individual user. It can be shared among users even though it is created locally. Because this DSN is file-based, it contains all information that is required to connect to a data source.

You can create multiple DSNs in this way and then reference them in your PC-based SAS/ACCESS Interface to ODBC code.

When you use the ODBC Data Source Administrator on the PC to create your ODBC data sources, the ODBC drivers for the particular databases that you want to access are often in the list of available drivers. This is especially true for drivers for the more common databases. If the ODBC driver that you want is not listed, you must work to obtain one.

**ODBC on a UNIX Platform**

ODBC on UNIX works a bit differently. The ODBC driver manager and ODBC drivers on the PC are available by default, so you need only plug them in. Because these components are not generally available on UNIX, you must instead work with third-party vendors to obtain them.

When you submit SAS/ACCESS Interface to ODBC code, SAS looks first for an ODBC driver manager. It checks the directories that are listed in such environment variables settings as `LD_LIBRARY_PATH`, `LIBPATH`, or `SHLIB_PATH`, depending on your UNIX platform. It uses the first ODBC driver manager that it finds.

The ODBC driver manager then checks INI files—either a stand-alone ODBC.INI file, or a combination of ODBC.INI and ODBCINST.INI files—for the DSNs that you specified in your code. To make sure that the intended INI files are referenced, you can use such environment variables settings as `ODB CinI` or `ODBCSYSINI`, depending on how your INI files are set up. You can set up global INI files for all your users, or you can set up INI files for single users or groups of users. This is similar to using the ODBC Data Source Administrator to create either SYSTEM or USER DSNs for PC platforms.

One or more INI files include a section for each DSN, and each section includes specific connection information for each data source from which you want to enable access to data. Some ODBC driver vendors provide tools with which you can build one or more of your INI files. However, editing a sample generic INI file that is provided with the ODBC driver is often done manually.

Most database vendors (such as Sybase, Oracle, or DB2) include ODBC drivers for UNIX platforms. To use SAS/ACCESS Interface to ODBC, pair an ODBC driver manager that is based in UNIX with your ODBC driver that is also based in UNIX. Freeware ODBC driver managers for UNIX such as unixODBC are generally available for download. Another alternative is to obtain the required ODBC client components for UNIX platforms from third-party vendors who market both ODBC drivers for databases and an ODBC driver manager that works with these drivers. To use SAS/ACCESS Interface to ODBC, you can select any ODBC client solution that you want as long as it is ODBC-compliant.
**ODBC for PC and UNIX Platforms**

These concepts are common across both PC and UNIX platforms.

- ODBC uses SQL syntax for queries and statement execution, or for statements that are executed as commands. However, all databases that support ODBC are not necessarily SQL databases. For example, many databases do not have system tables. Also, the term table can describe a variety of items—including a file, a part of a file, a group of files, a typical SQL table, generated data, or any potential source of data. This is an important distinction. All ODBC data sources respond to a base set of SQL statements such as SELECT, INSERT, UPDATE, DELETE, CREATE, and DROP in their simplest forms. However, some databases do not support other statements and more complex forms of SQL statements.

- The ODBC standard allows for various levels of conformance that is generally categorized as low, medium, and high. As previously mentioned, the level of SQL syntax that is supported varies. Also, some driver might not support many programming interfaces. SAS/ACCESS Interface to ODBC works with API calls that conform to the lowest level of ODBC compliance, Level 1. However, it does use some Level 2 API calls if they are available.

  SAS programmers or end users must make sure that their particular ODBC driver supports the SQL syntax to be used. If the driver supports a higher level of API conformance, some advanced features are available through the PROC SQL CONNECT statement and special queries that SAS/ACCESS Interface to ODBC supports. For more information, see “Special Catalog Queries” on page 741.

- The ODBC manager and drivers return standard operation states and custom text for any warnings or errors. The state variables and their associated text are available through the SAS SYSDBRC and SYSDBMSG macro variables.

**Key Considerations When Using ODBC Drivers and SAS on UNIX**

SAS/ACCESS Interface to ODBC on UNIX allows SAS customers to surface data from a wide variety of external data sources. Many customers using SAS on UNIX have had success using SAS/ACCESS Interface to ODBC with their ODBC client setups. These setups consist of an ODBC driver manager and ODBC drivers for the specific data sources to which customers need access. Critical to this success are the quality and completeness of third-party ODBC client components on UNIX that customers have chosen to use.

To maximize your chances of success, your ODBC driver must comply with the ODBC 3.5 (or later) specification. It must also but support the call sequences that SAS/ACCESS Interface to ODBC sends to the driver. Specifically, your ODBC driver manager and ODBC driver must support these ODBC calls:

```
SQLAllocConnect  SQLFreeStmt
SQLAllocEnv      SQLGetConnectAttr
SQLAllocHandle   SQLGetConnectOption
SQLAllocStmt     SQLGetCursorName
SQLBindCol       SQLGetDiagRec
SQLBindParameter  SQLGetFunctions
SQLBulkOperations SQLGetInfo
SQLCancel        SQLGetStmtAttr
SQLColAttribute   SQLGetStmtOption
SQLColumnPrivileges SQLGetTypeInfo
SQLColumns       SQLMoreResults
```
SQLConnect  SQLNumResultCols
SQLDataSources  SQLPrepare
SQLDescribeCol  SQLPrepareW
SQLDescribeColW  SQLPrimaryKeys
SQLDisconnect  SQLProcedureColumns
SQLDriverConnect  SQLProcedures
SQLEndTran  SQLRowCount
SQLExecDirect  SQLSetConnectAttr
SQLExecDirectW  SQLSetConnectOption
SQLExecute  SQLSetEnvAttr
SQLExtendedFetch  SQLSetPos
SQLFetch  SQLSetStmtAttr
SQLFetchScroll  SQLSetStmtOption
SQLForeignKeys  SQLSpecialColumns
SQLFreeConnect  SQLStatistics
SQLFreeEnv  SQLTablePrivileges
SQLFreeHandle  SQLTables

SAS/ACCESS Interface to ODBC sends a sequence of ODBC calls to the ODBC driver that you have chosen. The types and sequence in which these calls are made are compliant with the ODBC specification. If your ODBC driver fails to return the correct result or fails to work with SAS/ACCESS Interface to ODBC, here are your options.

- Make sure that you are running the current versions of your ODBC client components.

- Try to connect using a query tool that is not SAS. Most third-party ODBC driver and driver manager sources include such a query tool with their offerings. However, keep in mind that in some cases you might be able to connect using a query tool (that is not SAS) but not with SAS/ACCESS Interface to ODBC. This is because SAS calls might make a wider range of ODBC calls than an ODBC query tool that is not SAS would make.

- SAS Technical Support offers additional tools that can help you identify the root of your ODBC-related client problems and subsequently debug them.

- You can address some ODBC client issues by using certain SAS/ACCESS Interface to ODBC options or alternative engines to SAS/ACCESS Interface to ODBC.

- Once you have determined that your ODBC client issues are not related to SAS, you need to report your debugging results to your ODBC client providers. If you received your ODBC client components from a commercial ODBC driver vendor, you can work through that vendor’s technical support. If you use freeware or open-source ODBC client components—where formal technical support is not always available—your only recourse might be to communicate with the freeware user community.

SAS has not validated all ODBC drivers on the market and therefore makes no claims of certification or support.
LIBNAME Statement Specifics for ODBC

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to ODBC supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing ODBC.

`LIBNAME libref odbc <connection-options> <LIBNAME-options>;`

Arguments

`libref`  
specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

`odbc`  
specifies the SAS/ACCESS engine name for the ODBC interface.

`connection-options`  
provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to ODBC in many different ways. Specify only one of these methods for each connection because they are mutually exclusive.

- USER=, PASSWORD=, DATASRC=
- COMPLETE=
- NOPROMPT=
- PROMPT=
- READBUFF=
- REQUIRED=

Here is how these options are defined.

`USER=<'ODBC-user-name'>`  
lets you connect to an ODBC database with a user ID that is different from the default ID. USER= is optional.

Alias: UID=

`PASSWORD=<'ODBC-password'>`  
specifies the ODBC password that is associated with your user ID. PASSWORD= is optional. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you do not want to enter your DB2 password in uncoded text on this statement, see PROC PWENCODE in the Base SAS Procedures Guide for a method to encode it.

Alias: PWD=

`DATASRC=<'ODBC-data-source'>`  
specifies the ODBC data source to which you want to connect. For PC platforms, data sources must be configured by using the ODBC icon in the Windows
Control Panel. For UNIX platforms, data sources must be configured by modifying the .odbc.ini file. DSN= is an alias for this option that indicates that the connection is attempted using the ODBC SQLConnect API, which requires a data source name. You can also use a user ID and password with DSN=. If you want to use an ODBC file DSN, instead of supplying DATASRC='ODBC-data-source', use the PROMPT= or NOPROMPT= option, followed by "filedsn=(name-of-your-file-dsn);". For example:

```
libname mydblib odbc noprompt="filedsn=d:\share\msafiledsn.dsn;";
```

Alias: DATABASE=, DB=, DSN=

**COMPLETE=** specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMSG macro variable. If you do not specify enough correct connection options, you are prompted with a dialog box that displays the values from the COMPLETE= connection string. You can edit any field before you connect to the data source. This option is not supported on UNIX platforms. See your ODBC driver documentation for more details.

**NOPROMPT=** specifies connection options for your data source or database. Separate multiple options with a semicolon. If you do not specify enough correct connection options, an error is returned. No dialog box is displayed to help you complete the connection string.

**PROMPT=** specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMSG macro variable. PROMPT= does not immediately try to connect to the DBMS. A dialog box is displayed instead that contains the values that you entered in the PROMPT= connection string. You can edit values or enter additional values in any field before you connect to the data source.

Restriction: not supported on UNIX platforms

**READBUFF= number-of-rows**

Use this argument to improve the performance of most queries to ODBC. By setting the value of the READBUFF= argument in your SAS programs, you can find the optimal number of rows for a specified query on a specified table. The default buffer size is one row per fetch. The maximum is 32,767 rows per fetch, although a practical limit for most applications is less and depends on the available memory.

Alias: ROWSET_SIZE=

**REQUIRED=** specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMSG macro variable. If you do not specify enough correct connection options, a dialog box prompts you for the connection options. REQUIRED= lets you modify only required fields in the dialog box.

Restriction: not supported on UNIX platforms

See your ODBC driver documentation for a list of the ODBC connection options that your ODBC driver supports.

The following ODBC connection options are not supported on UNIX:
LIBNAME-options define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to ODBC, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

Table 26.1 SAS/ACCESS LIBNAME Options for ODBC

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>data-source specific</td>
</tr>
<tr>
<td>BL_LOG=</td>
<td>none</td>
</tr>
<tr>
<td>BL_OPTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>UNIQUE when data source supports only one cursor per connection, SHAREDREAD otherwise</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>FORWARDONLY</td>
</tr>
<tr>
<td>DATETIME2=</td>
<td>NO</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 when inserting rows, 0 when updating rows</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT_SQL=</td>
<td>data-source specific</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>based on row length</td>
</tr>
<tr>
<td>KEYSET_SIZE=</td>
<td>0</td>
</tr>
<tr>
<td>LOGIN_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for ODBC”</td>
</tr>
<tr>
<td>PRESERVE_TAB NAMES=</td>
<td>see “Naming Conventions for ODBC”</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>none</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the ODBC Interface”)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>0</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the ODBC Interface”)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE_SQL=</td>
<td>driver-specific</td>
</tr>
<tr>
<td>USE_ODBC_CL=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>

**ODBC LIBNAME Statement Examples**

In this example, USER=, PASSWORD=, and DATASRC= are connection options.

```sas
libname mydblib odbc user=myusr1 password=mypwd1 datasrc=mydatasource;
```

In this next example, the libref MYLIB uses the ODBC engine to connect to an Oracle database. The connection options are USER=, PASSWORD=, and DATASRC=.

```sas
libname mydblib odbc datasrc=mydatasourcemydatasource user=myusr1 password=mypwd1;
```

```sas
proc print data=mydblib.customers;
  where state='CA';
run;
```

In the next example, the libref MYDBLIB uses the ODBC engine to connect to a Microsoft SQL Server database. The connection option is NOPROMPT=.

```sas
libname mydblib odbc
  noprompt="uid=myusr1;pwd=mypwd1;dsn=sqlservr;"
  stringdates=yes;
```

```sas
proc print data=mydblib.customers;
  where state='CA';
run;
```
Data Set Options for ODBC

All SAS/ACCESS data set options in this table are supported for ODBC. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

Table 26.2  SAS/ACCESS Data Set Options for ODBC

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BULKLOAD=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DATETIME2=</td>
<td>NO</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for ODBC”</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARAM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
</tbody>
</table>
Option | Default Value
--- | ---
DBTYPE= | see “Data Types for ODBC”
ERRLIMIT= | 1
IGNORE_READ_ONLY_COLUMNS= | NO
INSERT_SQL= | LIBNAME option setting
INSERTBUFF= | LIBNAME option setting
KEYSET_SIZE= | LIBNAME option setting
NULLCHAR= | SAS
NULLCHARVAL= | a blank character
PRESERVE_COL_NAMES= | LIBNAME option setting
QUALIFIER= | LIBNAME option setting
QUERY_TIMEOUT= | LIBNAME option setting
READ_ISOLATION_LEVEL= | LIBNAME option setting
READ_LOCK_TYPE= | LIBNAME option setting
READBUFF= | LIBNAME option setting
SASDATEFMT= | none
SCHEMA= | LIBNAME option setting
UPDATE_ISOLATION_LEVEL= | LIBNAME option setting
UPDATE_LOCK_TYPE= | LIBNAME option setting
UPDATE_SQL= | LIBNAME option setting

**SQL Pass-Through Facility Specifics for ODBC**

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the ODBC interface.
• The `dbms-name` is `ODBC`.

• The `CONNECT` statement is required.

• `PROC SQL` supports multiple connections to ODBC. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default `odbc` alias is used. The functionality of multiple connections to the same ODBC data source might be limited by the particular data source driver.

• The `CONNECT` statement `database-connection-arguments` are identical to its `LIBNAME connection-options`. Not all ODBC drivers support all of these arguments. See your driver documentation for more information.

• On some DBMSs, the `DBMS-SQL-query` argument can be a DBMS-specific SQL EXECUTE statement that executes a DBMS stored procedure. However, if the stored procedure contains more than one query, only the first query is processed.

• These options are available with the `CONNECT` statement.
  • `AUTOCOMMIT`=
  • `CURSOR_TYPE`=
  • `KEYSET_SIZE`=
  • `QUERY_TIMEOUT`=
  • `READBUFF`=
  • `READ_ISOLATION_LEVEL`=
  • `TRACE`=
  • `TRACEFILE`=
  • `USE_ODBC_CL`=
  • `UTILCONN_TRANSIENT`=

**CONNECT Statement Examples**

These examples use ODBC to connect to a data source that is configured under the data source name `User's Data` using the alias `USER1`. The first example uses the connection method that is guaranteed to be present at the lowest level of ODBC conformance. `DATASRC=` names can contain quotation marks and spaces.

```
proc sql;
  connect to ODBC as user1
    (datasrc="User's Data" user=myusr1 password=mypwd1);
```

This example uses the connection method that represents a more advanced level of ODBC conformance. It uses the input dialog box that is provided by the driver. The `DATASRC=` and `USER=` arguments are within the connection string. The SQL pass-through facility therefore does not parse them but instead passes them to the ODBC manager.

```
proc sql;
  connect to odbc as user1
    (required="dsn=User's Data;uid=myusr1");
```

This example enables you to select any data source that is configured on your machine. The example uses the connection method that represents a more advanced level of ODBC conformance, Level 1. When connection succeeds, the connection string is
returned in the SQLXMSG and SYSDBMSG macro variables. The connection string can be stored if this method is used to configure a connection for later use.

```
proc sql;
  connect to odbc {required};
```

This next example prompts you to specify the information that is required to make a connection to the DBMS. You are prompted to supply the data source name, user ID, and password in the dialog boxes that are displayed.

```
proc sql;
  connect to odbc {prompt};
```

### Connection to Component Examples

This example sends an Oracle SQL query (presented in highlighted text) to the Oracle database for processing. The results from the query serve as a virtual table for the PROC SQL FROM clause. In this example MYCON is a connection alias.

```
proc sql;
  connect to odbc as mycon
    (datasrc=mysrv1 user=myusr1 password=mypwd1);

  select *
    from connection to mycon
      (select empid, lastname, firstname, hiredate, salary
       from sasdemo.employees
       where hiredate>='31.12.1988')
;
  disconnect from mycon;
  quit;
```

This next example gives the previous query a name and stores it as the SQL view Samples.Hires88. The CREATE VIEW statement appears highlighted.

```
libname samples 'SAS-library';

proc sql;
  connect to odbc as mycon
    (datasrc=mysrv1 user=myusr1 password=mypwd1);

  create view samples.hires88 as
    select *
      from connection to mycon
        (select empid, lastname, firstname, hiredate, salary
         from sasdemo.employees
         where hiredate>='31.12.1988');
  disconnect from mycon;
  quit;
```

This example connects to Microsoft Access and creates a view NEWORDERS from all columns in the ORDERS table.

```
proc sql;
  connect to odbc as mydb
    (datasrc=MSAccess?);
  create view neworders as
```
This next example sends an SQL query to Microsoft SQL Server, configured under the data source name SQL Server, for processing. The results from the query serve as a virtual table for the PROC SQL FROM clause.

```sql
proc sql;
  connect to odbc as mydb
  (datasrc="SQL Server" user=myusr1 password=mypwd1);
  select * from connection to mydb
  (select CUSTOMER, NAME, COUNTRY
   from CUSTOMERS
   where COUNTRY <> 'USA');
quit;
```

This example returns a list of the columns in the CUSTOMERS table.

```sql
proc sql;
  connect to odbc as mydb
  (datasrc="SQL Server" user=myusr1 password=mypwd1);
  select * from connection to mydb
  (ODBC::SQLColumns (), , "CUSTOMERS");
quit;
```

**Special Catalog Queries**

SAS/ACCESS Interface to ODBC supports the following special queries. Many databases provide or use system tables that allow queries to return the list of available tables, columns, procedures, and other useful information. ODBC provides much of this functionality through special application programming interfaces (APIs) to accommodate databases that do not follow the SQL table structure. You can use these special queries on SQL and non-SQL databases.

Here is the general format of the special queries:

```
ODBC::SQLAPI "parameter 1","parameter n"
```

- **ODBC::** required to distinguish special queries from regular queries.
- **SQLAPI** is the specific API that is being called. Neither ODBC:: nor SQLAPI are case sensitive.
- "parameter n" a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters; the underscore represents any single character. Each driver also has an escape character that can be used to place characters within the string. See the driver documentation to determine the valid escape character.

The values for the special query arguments are DBMS-specific. For example, you supply the fully qualified table name for a “Catalog” argument. In dBase, the value of “Catalog” might be `c:\dbase\tst.dbf` and in SQL Server, the value might be `test.customer`. In addition, depending on the DBMS that you are using, valid values
for a “Schema” argument might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all arguments within a parameter, use a comma to indicate the omitted arguments. If you do not specify any parameters, commas are not necessary. Special queries are not available for all ODBC drivers.

ODBC supports these special queries:

ODBC::SQLColumns <"Catalog", "Schema", "Table-name", "Column-name"> returns a list of all columns that match the specified arguments. If no arguments are specified, all accessible column names and information are returned.

ODBC::SQLColumnPrivileges <"Catalog", "Schema", "Table-name", "Column-name"> returns a list of all column privileges that match the specified arguments. If no arguments are specified, all accessible column names and privilege information are returned.

ODBC::SQLDataSources returns a list of database aliases to which ODBC is connected.

ODBC::SQLDBMSInfo returns a list of DB2 databases (DSNs) to which ODBC is connected. It returns one row with two columns that describe the DBMS name (such as SQL Server or Oracle) and the corresponding DBMS version.

ODBC::SQLForeignKeys <"PK-catalog", "PK-schema", "PK-table-name", "FK-catalog", "FK-schema", "FK-table-name"> returns a list of all columns that comprise foreign keys that match the specified arguments. If no arguments are specified, all accessible foreign key columns and information are returned.

ODBC::SQLGetTypeInfo returns information about the data types that are supported in the data source.

ODBC::SQLPrimaryKeys <"Catalog", "Schema", "Table-name"> returns a list of all columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If no table name is specified, this special query fails.

ODBC::SQLProcedures <"Catalog", "Schema", "Procedure-name"> returns a list of all procedures that match the specified arguments. If no arguments are specified, all accessible procedures are returned.

ODBC::SQLProcedureColumns <"Catalog", "Schema", "Procedure-name", "Column-name"> returns a list of all procedure columns that match the specified arguments. If no arguments are specified, all accessible procedure columns are returned.

ODBC::SQLSpecialColumns <"Identifier-type", "Catalog-name", "Schema-name", "Table-name", "Scope", "Nullable"> returns a list of the optimal set of columns that uniquely identify a row in the specified table.

ODBC::SQLStatistics <"Catalog", "Schema", "Table-name"> returns a list of the statistics for the specified table name. You can set the SQL_INDEX_ALL and SQL_ENSURE options in the SQLStatistics API call. If the table name argument is not specified, this special query fails.

ODBC::SQLTables <"Catalog", "Schema", "Table-name", "Type"> returns a list of all tables that match the specified arguments. If no arguments are specified, all accessible table names and information are returned.
Autopartitioning Scheme for ODBC

Overview

Autopartitioning for SAS/ACCESS Interface to ODBC is a modulo (MOD) function method. For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

Autopartitioning Restrictions

SAS/ACCESS Interface to ODBC places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here is how columns are partitioned.

- SQL_INTEGER, SQL_BIT, SQL_SMALLINT, and SQL_TINYINT columns are given preference.
- You can use SQL_DECIMAL, SQL_DOUBLE, SQL_FLOAT, SQL_NUMERIC, and SQL_REAL columns for partitioning under these conditions:
  - The ODBC driver supports converting these types to SQL_INTEGER by using the INTEGER cast function.
  - The precision minus the scale of the column is greater than 0 but less than 10—that is, \(0 < (\text{precision-scale}) < 10\).

The exception to the above rule is for Oracle SQL_DECIMAL columns. As long as the scale of the SQL_DECIMAL column is 0, you can use the column as the partitioning column.

Nullable Columns

If you select a nullable column for autopartitioning, the OR<column-name>IS NULL SQL statement is appended at the end of the SQL code that is generated for the threaded Read. This ensures that any possible NULL values are returned in the result set. Also, if the column to be used for the partitioning is SQL_BIT, the number of threads are automatically changed to two, regardless of how the DBSLICEPARM= option is set.

Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in the WHERE clause. For example, the following DATA step could not use a threaded Read to retrieve the data because all numeric columns in the table are in the WHERE clause.

data work.locemp;
  set trlib.MYEMPS;
  where EMPWUM<30 and ISTENURE=0 and
    SALARY<35000 and NUMCLASS>2;

Using DBSLICEPARM=

SAS/ACCESS Interface to ODBC defaults to three threads when you use autopartitioning. However, do not specify a maximum number of threads in DBSLICEPARM= to use for the threaded Read.

Using DBSLICE=

You might achieve the best possible performance when using threaded Reads by specifying the DBSLICE= option for ODBC in your SAS operation. This is especially true if your DBMS supports multiple database partitions and provides a mechanism to allow connections to individual partitions. If your DBMS supports this concept, you can configure an ODBC data source for each partition. You can also use the DBSLICE= clause to specify both the data source and the WHERE clause for each partition, as shown in this example.

```sas
proc print data=trilib.MYEMPS(DBSLICE=(DSN1="EMPNUM BETWEEN 1 AND 33" DSN2="EMPNUM BETWEEN 34 AND 66" DSN3="EMPNUM BETWEEN 67 AND 100"));
run;
```

You can also see "Configuring SQL Server Partitioned Views for Use with DBSLICE=" on page 744 for an example of configuring multiple partition access to a table.

Using the DATASOURCE= syntax is not required to use DBSLICE= with threaded Reads for the ODBC interface. The methods and examples described in DBSLICE= work well in cases where the table that you want to read is not stored in multiple partitions in your DBMS. These methods also give you flexibility in column selection. For example, if you know that the STATE column in your employee table only contains a few distinct values, you can customize your DBSLICE= clause accordingly.

```sas
datawork.locemp;
set trlib2.MYEMP(DBSLICE=('STATE='FL'" 'STATE='GA' " 'STATE='SC' " 'STATE='VA' " 'STATE='NC'"));
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

Configuring SQL Server Partitioned Views for Use with DBSLICE=

Microsoft SQL Server implements multiple partitioning by creating a global view across multiple instances of a Microsoft SQL Server database. For this example, assume that Microsoft SQL Server has been installed on three separate machines (SERVER1, SERVER2, SERVER3). Three ODBC data sources (SSPART1, SSPART2, SSPART3) have been configured against these servers. Also, a linked server definition for each of these servers has been defined. This example uses SAS to create the tables and associated views, but you can create the tables and associated outside of the SAS environment.

1. Create a local SAS table to build the Microsoft SQL Server tables.

```sas
data work.MYEMPS;
format HIREDATE mmddyy 0. SALARY 9.2
```
NUMCLASS 6. GENDER $1. STATE $2. EMPNUM 10.;
do EMPNUM=1 to 100;
morf=mod(EMPNUM,2)+1;
if(morf eq 1) then
   GENDER='F';
else
   GENDER='M';
SALARY=(ranuni(0)*5000);
HIREDATE=int(ranuni(13131)*3650);
whatstate=int(EMPNUM/5);
if(whatstate eq 1) then
   STATE='FL';
if(whatstate eq 2) then
   STATE='GA';
if(whatstate eq 3) then
   STATE='SC';
if(whatstate eq 4) then
   STATE='VA';
else
   STATE='NC';
ISTENURE=mod(EMPNUM,2);
NUMCLASS=int(EMPNUM/5)+2;
output;
end;
run;

Create a table on each of the SQL server databases with the same table structure, and insert one-third of the overall data into each table.

libname trlib odbc user=ssuser pw=sspwd dsn=sspart1;
libname trlib odbc user=ssuser pw=sspwd dsn=sspart2;
libname trlib odbc user=ssuser pw=sspwd dsn=sspart3;
proc datasets library=trlib;
delete MYEMPS1;run;
delete MYEMPS2;run;
delete MYEMPS3;run;
data trlib.MYEMPS1(drop=morf whatstate
   DBTYPE=(HIREDATE="datetime" SALARY="numeric(8,2)"
   NUMCLASS="smallint" GENDER="char(1)" ISTENURE="bit" STATE="char(2)"
   EMPNUM="int NOT NULL Primary Key CHECK (EMPNUM BETWEEN 0 AND 33)"));
set work.MYEMPS;
where (EMPNUM BETWEEN 0 AND 33);
run;
data trlib.MYEMPS2(drop=morf whatstate
   DBTYPE=(HIREDATE="datetime" SALARY="numeric(8,2)"
   NUMCLASS="smallint" GENDER="char(1)" ISTENURE="bit" STATE="char(2)"
   EMPNUM="int NOT NULL Primary Key CHECK (EMPNUM BETWEEN 34 AND 66)"));
set work.MYEMPS;
where (EMPNUM BETWEEN 34 AND 66);
run;
data trlib.MYEMPS3(drop=morf whatstate
   DBTYPE=(HIREDATE="datetime" SALARY="numeric(8,2)"
NUMCLASS="smallint" GENDER="char(1)" ISTENURE="bit" STATE="char(2)"
EMPNUM="int NOT NULL Primary Key CHECK (EMPNUM BETWEEN 67 AND 100)"
set work.MYEMPS;
where (EMPNUM BETWEEN 67 AND 100);
run;

These table definitions also use CHECK constraints to enforce the distribution of the
data on each of the subtables of the target view.

3. Create a view using the UNION ALL construct on each Microsoft SQL Server
instance that references the other two tables.

/*SERVER1,SSPART1*/
proc sql noerrorstop;
connect to odbc (UID=ssuser PWD=sspwd DSN=SSPART1);
execute (drop view MYEMPS) by odbc;
execute (create view MYEMPS AS
      SELECT * FROM users.ssuser.MYEMPS1
      UNION ALL
      SELECT * FROM SERVER2.users.ssuser.MYEMPS2
      UNION ALL
      SELECT * FROM SERVER3.users.ssuser.MYEMPS3) by odbc;
quit;

/*SERVER2,SSPART2*/
proc sql noerrorstop;
connect to odbc (UID=ssuser PWD=sspwd DSN=SSPART2);
execute (drop view MYEMPS) by odbc;
execute (create view MYEMPS AS
      SELECT * FROM users.ssuser.MYEMPS2
      UNION ALL
      SELECT * FROM SERVER1.users.ssuser.MYEMPS1
      UNION ALL
      SELECT * FROM SERVER3.users.ssuser.MYEMPS3) by odbc;
quit;

/*SERVER3,SSPART3*/
proc sql noerrorstop;
connect to odbc (UID=ssuser PWD=sspwd DSN=SSPART3);
execute (drop view MYEMPS) by odbc;
execute (create view MYEMPS AS
      SELECT * FROM users.ssuser.MYEMPS3
      UNION ALL
      SELECT * FROM SERVER2.users.ssuser.MYEMPS2
      UNION ALL
      SELECT * FROM SERVER1.users.ssuser.MYEMPS1) by odbc;
quit;

This creates a global view that references the entire data set.

4. Set up your SAS operation to perform the threaded Read.

proc print data=trlib.MYEMPS(DBLICE=(sspart1="EMPNUM BETWEEN 1 AND 33"
sspart2="EMPNUM BETWEEN 34 AND 66"
sspart3="EMPNUM BETWEEN 67 AND 100");
run;

The DBSLICE= option contains the Microsoft SQL Server partitioning information.
This configuration lets the ODBC interface access the data for the MYEMPS view directly from each subtable on the corresponding Microsoft SQL Server instance. The data is inserted directly into each subtable, but this process can also be accomplished by using the global view to divide up the data. For example, you can create empty tables and then create the view as seen in the example with the UNION ALL construct. You can then insert the data into the view MYEMPS. The CHECK constraints allow the Microsoft SQL Server query processor to determine which subtables should receive the data.

Other tuning options are available when you configure Microsoft SQL Server to use partitioned data. For more information, see the "Creating a Partitioned View" and "Using Partitioned Views" sections in Creating and Maintaining Databases (SQL Server 2000).

### DBLOAD Procedure Specifics for ODBC

**Overview**

For general information about this feature, see the Appendix 3, “DBLOAD Procedure.” ODBC examples are available.

SAS/ACCESS Interface to ODBC supports all DBLOAD procedure statements (except ACCDESC=) in batch mode. Here are the DBLOAD procedure specifics for ODBC:

- The DBLOAD step DBMS= value is **ODBC**.
- Here are the database description statements that PROC DBLOAD uses:
  - **DSN=** '<ODBC-data-source>'; specifies the name of the data source in which you want to store the new ODBC table. The data-source is limited to eight characters.
    - The data source that you specify must already exist. If the data source name contains the _, $, @, or # special character, you must enclose it in quotation marks. The ODBC standard recommends against using special characters in data source names, however.
  - **USER=** '<ODBC-user name>'; lets you connect to an ODBC database with a user ID that is different from the default ID. USER= is optional in ODBC. If you specify USER=, you must also specify PASSWORD=.. If USER= is omitted, your default user ID is used.
  - **PASSWORD=** '<ODBC-password>'; specifies the ODBC password that is associated with your user ID.
    - PASSWORD= is optional in ODBC because users have default user IDs. If you specify USER=, you must specify PASSWORD=.
      - **Note:** If you do not want to enter your ODBC password in uncoded text on this statement, see PROC PWENCODE in the Base SAS Procedures Guide for a method to encode it.
  - **BULKCOPY=** YES|NO; determines whether SAS uses the Microsoft Bulk Copy facility to insert data into a DBMS table (Microsoft SQL Server only). The default value is NO.
    - The Microsoft Bulk Copy (BCP) facility lets you efficiently insert rows of data into a DBMS table as a unit. As the ODBC interface sends each row of data to BCP, the data is written to an input buffer. When you have inserted all rows or the buffer reaches a certain size, all rows are inserted as a unit into the table, and...
the data is committed to the table. The DBCOMMIT= data set option determines the size of the buffer.

You can also set the DBCOMMIT=n option to commit rows after every n insertions.

If an error occurs, a message is written to the SAS log, and any rows that have been inserted in the table before the error are rolled back.

**Note:** BULKCOPY= is not supported on UNIX.

- Here is the TABLE= statement:

  ```
  TABLE= <authorization-id.>table-name;
  ```

  identifies the table or view that you want to use to create an access descriptor.

  The TABLE= statement is required.

  The authorization-id is a user ID or group ID that is associated with the table.

- Here is the NULLS statement:

  ```
  NULLS variable-identifier-1=Y|N|D < . . . variable-identifier-n=Y|N|D >;
  ```

  enables you to specify whether the columns that are associated with the listed SAS variables allow NULL values. By default, all columns accept NULL values.

  The NULLS statement accepts any one of these three values:

  - **Y** – specifies that the column accepts NULL values. This is the default.
  - **N** – specifies that the column does not accept NULL values.
  - **D** – specifies that the column is defined as NOT NULL WITH DEFAULT.

**Examples**

This example creates a new ODBC table, MYUSR1.EXCHANGE, from the DLIB.RATEOFEX data file. You must be granted the appropriate privileges in order to create new ODBC tables or views.

```sas
proc dbload dbms=odbc data=dlib.rateofex;
  dsn=sample;
  user='myusr1';
  password='mypwd1';
  table=exchange;
  rename fgnindol=fgnindollars
  4=dollarsinfgn;
  nulls updated=n fgnindollars=n
doollarsinfgn=n country=n;
load;
run;
```

This next example sends only an ODBC SQL GRANT statement to the SAMPLE database and does not create a new table. Therefore, the TABLE= and LOAD statements are omitted.

```sas
proc dbload dbms=odbc;
  user='myusr1';
  password='mypwd1';
  dsn=sample;
  sql grant select on myusr1.exchange
to testcase;
run;
```
Temporary Table Support for ODBC

SAS/ACCESS Interface to ODBC supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

Passing SAS Functions to ODBC

SAS/ACCESS Interface to ODBC passes the following SAS functions to the data source for processing if the DBMS server supports this function. Where the ODBC function name differs from the SAS SQL function name, the ODBC name appears in parentheses. For details, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

ABS  LOG10
ARCOS LOWCASE
ARSIN MAX
ATAN MIN
AVG  SIGN
CEIL  SIN
COS  SQRT
COT  STRIP
COUNT SUM
EXP  TAN
FLOOR  UPCASE
LOG

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to ODBC. Due to incompatibility in date and time functions between ODBC and SAS, ODBC might not process them correctly. Check your results to determine whether these functions are working as expected.

BYTE (CHAR)  REPEAT
COMPRESS (REPLACE)  SECOND
DATE (CURDATE)  SOUNDEX
DATEPART  SUBSTR (SUBSTRING)
DATETIME (NOW)  TIME (CURTIME)
DAY (DAYOFMONTH)  TIMEPART
HOUR  TODAY (CURDATE)
INDEX (LOCATE)  TRIMN (RTRIM)
LENGTH  TRANWRD (REPLACE)
MINUTE  WEEKDAY (DAYOFWEEK)
MONTH  YEAR
QTR (QUARTER)
Passing Joins to ODBC

For a multiple libref join to pass to ODBC, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- data source (DATASRC=)
- catalog (QUALIFIER=)
- update isolation level (UPDATE_ISOLATION_LEVEL=, if specified)
- read isolation level (READ_ISOLATION_LEVEL=, if specified)
- prompt (PROMPT=, must not be specified)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for ODBC

The BULKLOAD= on page 112 LIBNAME option calls the Bulk Copy (BCP) facility, which lets you efficiently insert rows of data into a DBMS table as a unit. BCP= is an alias for this option.

Windows Specifics

The Bulk Copy facility is available only when you are accessing Microsoft SQL Server data on Windows platforms. To use this facility, your installation of Microsoft SQL Server must include the ODBCBCP.DLL file. BULKCOPY= is not available on UNIX.

As the ODBC interface sends rows of data to the Bulk Copy facility, data is written to an input buffer. When you send all rows or when the buffer reaches a certain size (DBCOMMIT= determines this), all rows are inserted as a unit into the table and the data is committed to the table. You can also set DBCOMMIT= to commit rows after a specified number of rows are inserted.

If an error occurs, a message is written to the SAS log, and any rows that were inserted before the error are rolled back.

Locking in the ODBC Interface

The following LIBNAME and data set options let you control how the ODBC interface handles locking when a table is read. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

READ_LOCK_TYPE=ROW | TABLE | NOLOCK

ROW locks a row if any of its columns are accessed during a read transaction.

TABLE locks an entire table.
NOLOCK does not lock a DBMS table or any rows during a read transaction.

**UPDATE_LOCK_TYPE= ROW | TABLE | NOLOCK**

ROW locks a row if any of its columns are going to be updated.

TABLE locks an entire table.

NOLOCK does not lock a DBMS table or any rows when reading them for an update.

**READ_ISOLATION_LEVEL= S | RR | RC | RU | V**

The ODBC driver manager supports the S, RR, RC, RU, and V isolation levels that are defined in this table.

**Table 26.3 Isolation Levels for ODBC**

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (serializable)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads.</td>
</tr>
<tr>
<td>RR (repeatable read)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RC (read committed)</td>
<td></td>
</tr>
<tr>
<td>RU (read uncommitted)</td>
<td>Allows dirty Reads, nonrepeatable Reads, and phantom Reads.</td>
</tr>
<tr>
<td>V (versioning)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads. These transactions are serializable but higher concurrency is possible than with the serializable isolation level. Typically, a nonlocking protocol is used.</td>
</tr>
</tbody>
</table>

Here are how the terms in the table are defined.

**Dirty reads**

A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it can see changes that are made by those concurrent transactions even before they commit.

For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.

**Nonrepeatable reads**

If a transaction exhibits this phenomenon, it might read a row once and later fail when it attempts to read that row again in the same transaction. The row might have been changed or even deleted by a concurrent transaction. Therefore, the read is not necessarily repeatable.

For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T2 has now retrieved the same row twice but has seen two different values for it.

**Phantom reads**

When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.
For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist, a phantom.

**UPDATE_ISOLATION_LEVEL= S | RR | RC | V**

The ODBC driver manager supports the S, RR, RC, and V isolation levels defined in the preceding table.

---

### Naming Conventions for ODBC

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Because ODBC is an application programming interface (API) rather than a database, table names and column names are determined at run time. Most SAS names can be up to 32 characters long. SAS/ACCESS Interface to ODBC supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, SAS truncates them to 32 characters. If truncating a column name would result in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see *LIBNAME Statement for Relational Databases* on page 96.)

This example specifies Sybase as the DBMS.

```sas
libname mydblib odbc user=myusr1 password=mypwd1
database=sybase;
```

data mydblib.a;
  x=1;
  y=2;
run;

Sybase is generally case sensitive. This example would therefore produce a Sybase table named `a` with columns named `x` and `y`.

If the DBMS being accessed is not case sensitive, such as Oracle, the example would produce an Oracle table named `A` and columns named `X` and `Y`. The object names would be normalized to uppercase.

---

### Data Types for ODBC

#### Overview

Every column in a table has a name and a data type. The data type tells the DBMS how much physical storage to set aside for the column and the form in which the data is
stored. This section includes information about ODBC null and default values and data conversions.

**ODBC Null Values**

Many relational database management systems have a special value called NULL. A DBMS NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DBMS NULL value, it interprets it as a SAS missing value.

In most relational databases, columns can be defined as NOT NULL so that they require data (they cannot contain NULL values). When a column is defined as NOT NULL, the DBMS does not add a row to the table unless the row has a value for that column. When creating a DBMS table with SAS/ACCESS, you can use the `DBNULL=` data set option to indicate whether NULL is a valid value for specified columns.

ODBC mirrors the behavior of the underlying DBMS with regard to NULL values. See the documentation for your DBMS for information about how it handles NULL values.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how the DBMS handles SAS missing character values, use the `NULLCHAR=` and `NULLCHARVAL=` data set options.

**LIBNAME Statement Data Conversions**

This table shows all data types and default SAS formats that SAS/ACCESS Interface to ODBC supports. It does not explicitly define the data types as they exist for each DBMS. It lists the SQL types that each DBMS data type would map to. For example, a CHAR data type under DB2 would map to an ODBC data type of SQL_CHAR. All data types are supported.

<table>
<thead>
<tr>
<th>ODBC Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_CHAR</td>
<td>$w:</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
<td>$w:</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td>$w:</td>
</tr>
<tr>
<td>SQL_BINARY</td>
<td>$w:*</td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
<td>$w:*</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td>$w:*</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>w: or w:d or none if w and d are not specified</td>
</tr>
<tr>
<td>SQL_NUMERIC</td>
<td>w: or w:d or none if w and d are not specified</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>11:</td>
</tr>
<tr>
<td>ODBC Data Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
<td>6.</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td>4.</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>1.</td>
</tr>
<tr>
<td>SQL_REAL</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FLOAT</td>
<td>none</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>none</td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td>20.</td>
</tr>
<tr>
<td>SQL_INTERVAL</td>
<td>$w.</td>
</tr>
<tr>
<td>SQL_GUID</td>
<td>$w.</td>
</tr>
<tr>
<td>SQL_TYPE_DATE</td>
<td>DATE9.</td>
</tr>
<tr>
<td>SQL_TYPE_TIME</td>
<td>TIME8.</td>
</tr>
<tr>
<td></td>
<td>ODBC cannot support fractions of seconds for time values</td>
</tr>
<tr>
<td>SQL_TYPE_TIMESTAMP</td>
<td>DATETIME$w,d$ where $w$ and $d$ depend on precision</td>
</tr>
</tbody>
</table>

* Because the ODBC driver does the conversion, this field is displayed as if the $HEXw$ format were applied.

This table shows the default data types that SAS/ACCESS Interface to ODBC uses when creating tables. SAS/ACCESS Interface to ODBC lets you specify non-default data types by using the `DBTYPE=` data set option.

**Table 26.5 Default ODBC Output Data Types**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Default ODBC Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m.n$</td>
<td>SQL_DOUBLE or SQL_NUMERIC using $m.n$ if the DBMS allows it</td>
</tr>
<tr>
<td>$sn.$</td>
<td>SQL_VARCHAR using $n$</td>
</tr>
<tr>
<td>datetime formats</td>
<td>SQL_TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>SQL_DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>SQL_TIME</td>
</tr>
</tbody>
</table>
Introduction to SAS/ACCESS Interface to OLE DB

For available SAS/ACCESS features, see OLE DB supported features on page 86. For more information about OLE DB, see your OLE DB documentation.
Microsoft OLE DB is an application programming interface (API) that provides access to data that can be in a database table, an email file, a text file, or another type of file. This SAS/ACCESS interface accesses data from these sources through OLE DB data providers such as Microsoft Access, Microsoft SQL Server, and Oracle.

LIBNAME Statement Specifics for OLE DB

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to OLE DB supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing OLE DB.

```
LIBNAME libref oledb <connection-options> <LIBNAME-options>;
```

Arguments

- **libref**
  - specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

- **oledb**
  - specifies the SAS/ACCESS engine name for the OLE DB interface.

- **connection-options**
  - provide connection information and control how SAS manages the timing and concurrence of the connection to the data source. You can connect to a data source either by using OLE DB Services or by connecting directly to the provider. For details, see “Connecting with OLE DB Services” on page 761 and “Connecting Directly to a Data Provider” on page 761.

  These connection options are available with both connection methods. Here is how they are defined.

  - **USER=<'OLE-DB-user-name'>**
    - lets you connect to an OLE DB data source with a user ID that is different from the default ID. The default is your user ID.

  - **PASSWORD=<'OLE-DB-password'>**
    - specifies the OLE DB password that is associated with your user ID. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you do not wish to enter your OLE DB password in uncoded text, see PROC PWENCODE in the Base SAS Procedures Guide for a method to encode it.

  - **DATASOURCE=<'data-source'>**
    - identifies the data source object (such as a relational database server or a local file) to which you want to connect.

  - **PROVIDER=<'provider-name'>**
    - specifies which OLE DB provider to use to connect to the data source. This option is required during batch processing. There is no restriction on the length of the provider-name. If the provider-name contains blank spaces or special characters, enclose it in quotation marks. If you do not specify a provider, an OLE DB Services dialog box prompts you for connection information. In batch
mode, if you do not specify a provider the connection fails. To use the Ace, you must specify Microsoft.ACE.OLEDB.12.0. If you are using the Microsoft Jet OLE DB 4.0 provider, specify PROVIDER=JET.

**PROPERTIES=(<property-1>=<value-1>< . . . <property-n>=<value-n>>)***

This specifies standard provider properties that enable you to connect to a data source and define connection attributes. If a property name or value contains embedded spaces or special characters, enclose the name or value in quotation marks. Use a blank space to separate multiple properties. If your provider supports a password property, that value cannot be encoded. To use an encoded password, use the PASSWORD= option instead. See your provider documentation for a list and description of all properties that your provider supports. No properties are specified by default.

**PROVIDER_STRING=<extended-properties>***

This specifies provider-specific extended connection information, such as the file type of the data source. If the string contains blank spaces or special characters, enclose it in quotation marks. For example, the Ace provider accepts strings that indicate file type, such as 'Excel 8.0'. The following example uses the Ace provider to access the spreadsheet Y2KBUDGET.XLS. Specify the 'Excel 8.0' provider string so that Ace recognizes the file as an Excel 8.0 worksheet.

```
libname budget oledb provider='Microsoft.ACE.OLEDB.12.0'
provider_string='Excel 8.0' datasource='d:\excel80\Y2Kbudget.xls';
```

**OLEDB_SERVICES=**YES | NO

determines whether SAS uses OLE DB Services to connect to the data source. Specify YES to use OLE DB Services or specify NO to use the provider to connect to the data source. When you specify PROMPT=YES and OLEDB_SERVICES=YES, you can set more options than you would otherwise be able to set by being prompted by the provider's dialog box. If OLEDB_SERVICES=NO, you must specify PROVIDER= first so that the provider's prompt dialog boxes are used. If PROVIDER= is omitted, SAS uses OLE DB Services, even if you specify OLEDB_SERVICES=NO. YES is the default. For Microsoft SQL Server data, if BULKLOAD=YES, OLEDB_SERVICES= is set to NO. When OLEDB_SERVICES=YES and a successful connection is made, the complete connection string is returned in the SYSDBMS Macro variable.

**PROMPT =**YES | NO

determines whether one of these interactive dialog boxes is displayed to guide you through the connection process:

- an OLE DB provider dialog box if OLEDB_SERVICES=NO and you specify a provider.
- an OLE DB Services dialog box if OLEDB_SERVICES=YES or if you do not specify a provider.

Generally preferred over the provider's dialog box, the OLE DB Services dialog box lets you set options more easily. If you specify a provider and set OLEDB_SERVICES=NO, the default is PROMPT=NO. Otherwise, the default is PROMPT=YES. If OLEDB_SERVICES=YES or if you do not specify a provider, an OLE DB Services dialog box is displayed even if you specify PROMPT=NO. Specify no more than one of the following options in each LIBNAME statement: COMPLETE=, REQUIRED=, PROMPT=. Any properties that you specify in the PROPERTIES= option are displayed in the prompting interface, and you can edit any field.

**LIBNAME Statement Specifics for OLE DB**

757
UDL_FILE="path-and-file-name"

specifies the path and filename for a Microsoft universal data link (UDL). For example, you could specify

UDL_FILE=C:\WinNT\profiles\me\desktop\MyDBLink.UDL. This option does not support SAS filerefs. SYSDBMSG is not set on successful completion. For more information, see Microsoft documentation about the Data Link API. This option overrides any values that are set with the INIT_STRING=, PROVIDER=, and PROPERTIES= options.

This connection option is available only when you use OLE DB Services.

INIT_STRING='property-1=value-1<…;property-n=value-n>'

specifies an initialization string, enabling you to bypass the interactive prompting interface yet still use OLE DB Services. (This option is not available if OLEDB_SERVICES=NO.) Use a semicolon to separate properties. After you connect to a data source, SAS returns the complete initialization string to the macro variable SYSDBMSG, which stores the connection information that you specify in the prompting window. You can reuse the initialization string to make automated connections or to specify connection information for batch jobs. For example, assume that you specify this initialization string:

init_string='Provider=SQLOLEDB;Password=dbmgr1;Persist Security Info=True;User ID=rachel;Initial Catalog=users; Data Source=dwtsrv1';

Here is what the content of the SYSDBMSG macro variable would be:

OLEDB: Provider=SQLOLEDB;Password=dbmgr1;
Persist Security Info=True;User ID=rachel;Initial Catalog=users; Data Source=dwtsrv1;

If you store this string for later use, delete the OLEDB: prefix and any initial spaces before the first listed option. There is no default value. However, if you specify a null value for this option, the OLE DB Provider for ODBC (MSDASQL) is used with your default data source and its properties. See your OLE DB documentation for more information about these default values. This option overrides any values that are set with the PROVIDER= and PROPERTIES= options. To write the initialization string to the SAS log, submit this code immediately after connecting to the data source:

%put %superq(SYSDBMSG);

Only these connection options are available when you connect directly to a provider.

COMPLETE=YES | NO

specifies whether SAS attempts to connect to the data source without prompting you for connection information. If you specify COMPLETE=YES and the connection information that you specify in your LIBNAME statement is sufficient, then SAS makes the connection and does not prompt you for additional information. If you specify COMPLETE=YES and the connection information that you specify in your LIBNAME statement is not sufficient, the provider's dialog box prompts you for additional information. You can enter optional information as well as required information in the dialog box. NO is the default value. COMPLETE= is available only when you set OLEDB_SERVICES=NO and you specify a provider. It is not available in the SQL pass-through facility. Specify no more than one of these options in each LIBNAME statement: COMPLETE=, REQUIRED=, PROMPT=.

REQUIRED=YES | NO

specifies whether SAS attempts to connect to the data source without prompting you for connection information and whether you can interactively specify
optional connection information. If you specify REQUIRED=YES and the connection information that you specify in your LIBNAME statement is sufficient, SAS makes the connection and you are not prompted for additional information. If you specify REQUIRED=YES and the connection information that you specify in your LIBNAME statement is not sufficient, the provider's dialog box prompts you for the required connection information. You cannot enter optional connection information in the dialog box. NO is the default value. REQUIRED= is available only when you set OLEDB_SERVICES=NO and you specify a provider in the PROVIDER= option. It is not available in the SQL pass-through facility. Specify no more than one of these options in each LIBNAME statement: COMPLETE=, REQUIRED=, PROMPT=.

**LIBNAME-options**

define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to OLE DB, with the applicable default values. For details, see *LIBNAME Options for Relational Databases on page 101.*

**Table 27.1 SAS/ACCESS LIBNAME Options for OLE DB**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>data-source specific</td>
</tr>
<tr>
<td>BL_KEEPIDENTITY=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_KEEPNULLS=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_OPTIONS=</td>
<td>not specified</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>CELLPROP=</td>
<td>VALUE</td>
</tr>
<tr>
<td>COMMAND_TIMEOUT=</td>
<td>0 (no time–out)</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>FORWARD_ONLY</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEND_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT_SQL=</td>
<td>data-source specific</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>1</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>NONE</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for OLE DB”</td>
</tr>
<tr>
<td>PRESERVE_GUID=</td>
<td>on page 186</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>see “Naming Conventions for OLE DB”</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>none</td>
</tr>
<tr>
<td>QUALIFY_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>not set</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>1</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>not set (see “Locking in the OLE DB Interface”)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>see “Locking in the OLE DB Interface”</td>
</tr>
</tbody>
</table>
## Option Default Value

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>not set (see “Locking in the OLE DB Interface”)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>

### Connecting with OLE DB Services

By default, SAS/ACCESS Interface to OLE DB uses OLE DB services because this is often the fastest and easiest way to connect to a data provider.

OLE DB Services provides performance optimizations and scaling features, including resource pooling. It also provides interactive prompting for the provider name and connection information.

Assume that you submit a simple LIBNAME statement, such as this one:

```plaintext
libname mydblib oledb;
```

SAS directs OLE DB Services to display a dialog box that contains tabs where you can enter the provider name and connection information.

After you make a successful connection using OLE DB Services, you can retrieve the connection information and reuse it in batch jobs and automated connections. For more information, see the connection options INIT_STRING= and OLEDB_SERVICES=.

### Connecting Directly to a Data Provider

To connect to a data source, SAS/ACCESS Interface to OLE DB requires a provider name and provider-specific connection information such as the user ID, password, schema, or server name. If you know all of this information, you can connect directly to a provider without using OLE DB services.

If you are connecting to Microsoft SQL Server and you specify the SAS/ACCESS BULKLOAD=YES option, you must connect directly to the provider by specifying this information:

- the name of the provider (PROVIDER=)
- any required connection information
After you connect to your provider, you can use the special OLE DB `PROVIDER_INFO` query to make subsequent unprompted connections easier. You can submit this special query as part of a PROC SQL query to display all available provider names and properties. For an example, see “Examples of Special OLE DB Queries” on page 768.

If you know only the provider name and you are running an interactive SAS session, you can be prompted for the provider's properties. Specify `PROMPT=YES` to direct the provider to prompt you for properties and other connection information. Each provider displays its own prompting interface.

If you run SAS in a batch environment, specify only `USER=`, `PASSWORD=`, `DATASOURCE=`, `PROVIDER=`, and `PROPERTIES=`.

**OLE DB LIBNAME Statement Examples**

In this example, the libref MYDBLIB uses the SAS/ACCESS OLE DB engine to connect to a Microsoft SQL Server database.

```sas
libname mydblib oledb user=myusr1 password=mypwd1
datasource=dept203 provider=sqloledb properties=('initial catalog'=mgronly);
proc print data=mydblib.customers;
  where state='CA';
run;
```

In the next example, the libref MYDBLIB uses the SAS/ACCESS engine for OLE DB to connect to an Oracle database. Because prompting is enabled, you can review and edit the user, password, and data source information in a dialog box.

```sas
libname mydblib oledb user=myusr1 password=mypwd1 datasource=mydb.world
  provider=msdaora prompt=yes;
proc print data=mydblib.customers;
  where state='CA';
run;
```

For this next example, you submit a basic LIBNAME statement, so an OLE DB Services dialog box prompts you for the provider name and property values.

```sas
libname mydblib oledb;
```

The advantage of being prompted is that you do not need to know any special syntax to set the values for the properties. Prompting also enables you to set more options than you might when you connect directly to the provider (and do not use OLE DB Services).

---

**Data Set Options for OLE DB**

All SAS/ACCESS data set options in this table are supported for OLE DB. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

*Table 27.2  SAS/ACCESS Data Set Options for OLE DB*

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_KEEPIDENTITY=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>BL_KEEPNULLS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>BL_OPTIONS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>COMMAND_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>CURSOR_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td><em>ALL</em> =YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for OLE DB”</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for OLE DB”</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT_SQL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for OLE DB

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the OLE DB interface.

- The `dbms-name` is **OLEDB**.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to OLE DB. If you use multiple simultaneous connections, you must use an *alias* to identify the different connections. If you do not specify an alias, the default alias, **OLEDB**, is used. The functionality of multiple connections to the same OLE DB provider might be limited by a particular provider.
- The CONNECT statement `database-connection-arguments` are identical to its LIBNAME connection options. For some data sources, the connection options have default values and are therefore not required.

Not all OLE DB providers support all connection options. See your provider documentation for more information.

- Here are the LIBNAME options that are available with the CONNECT statement.
  - `AUTOCOMMIT=`
  - `CELLPROP=`

---

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>not set</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>YES</td>
</tr>
</tbody>
</table>

---
Examples

This example uses an alias to connect to a Microsoft SQL Server database and select a subset of data from the PAYROLL table. The SAS/ACCESS engine uses OLE DB Services to connect to OLE DB because this is the default action when the OLEDB_SERVICES= option is omitted.

```sql
proc sql;
connect to oledb as finance
  (user=myusr1 password=mypwd1 datasource=dwtsrv1
   provider=sqloledb);
select * from connection to finance (select * from payroll
   where jobcode='FA3');
quit;
```

In this example, the CONNECT statement omits the provider name and properties. An OLE DB Services dialog box prompts you for the connection information.

```sql
proc sql;
connect to oledb;
quit;
```

This example uses OLE DB Services to connect to a provider that is configured under the data source name User's Data with the alias USER1. Note that the data source name can contain quotation marks and spaces.

```sql
proc sql;
connect to oledb as user1
  (provider='Microsoft.ACE.OLEDB.12.0' datasource='c:\db1.mdb');
```

Special Catalog Queries

Overview

SAS/ACCESS Interface to OLE DB supports the following special queries. Many databases provide or use system tables that allow queries to return the list of available tables, columns, procedures, and other useful information. OLE DB provides much of this functionality through special application programming interfaces (APIs) to accommodate databases that do not follow the SQL table structure. You can use these special queries on SQL and non-SQL databases.

Not all OLE DB providers support all queries. See your provider documentation for more information.
Here is the general format of the special queries:

```
OLEDB::schema-rowset("parameter 1","parameter n")
```

OLEDDB::

is required to distinguish special queries from regular queries.

```
schema-rowset
```

is the specific schema rowset that is being called. All valid schema rowsets are listed under the IDBSchemaRowset Interface in the Microsoft OLE DB Programmer's Reference. Both OLEDDB:: and schema-rowset are case sensitive.

"parameter n"

is a quoted string that is enclosed by commas. The values for the special query arguments are specific to each data source. For example, you supply the fully qualified table name for a "Qualifier" argument. In dBase, the value of "Qualifier" might be `c:\\dbase\\tst.dbf`, and in SQL Server, the value might be `test.customer`. In addition, depending on the data source that you use, values for an "Owner" argument might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all arguments within a parameter, use commas to indicate omitted arguments. If you do not specify any parameters, no commas are necessary. These special queries might not be available for all OLE DB providers.

OLE DB supports these special queries:

```
OLEDB::ASSERTIONS( <"Catalog", "Schema", "Constraint-Name">)
returns assertions that are defined in the catalog that a given user owns.
```

```
OLEDB::CATALOGS( <"Catalog")
returns physical attributes that are associated with catalogs that are accessible from the DBMS.
```

```
OLEDB::CHARACTER_SETS( <"Catalog", "Schema","Character-Set-Name")
returns the character sets that are defined in the catalog that a given user can access.
```

```
OLEDB::CHECK_CONSTRAINTS(<"Catalog", "Schema", "Constraint-Name")
returns check constraints that are defined in the catalog and that a given user owns.
```

```
OLEDB::COLLATIONS(<"Catalog", "Schema", "Collation-Name")
returns the character collations that are defined in the catalog and that a given user can access.
```

```
OLEDB::COLUMN_DOMAIN_USAGE( <"Catalog", "Schema", "Domain-Name", "Column-Name")
returns the columns that are defined in the catalog, are dependent on a domain that is defined in the catalog, and a given user owns.
```

```
OLEDB::COLUMN_PRIVILEGES( <"Catalog", "Schema", "Table-Name", "Column-Name", "Grantor", "Grantee")
returns the privileges on columns of tables that are defined in the catalog that a given user grants or can access.
```

```
OLEDB::COLUMNS( <"Catalog", "Schema", "Table-Name", "Column-Name")
returns the columns of tables that are defined in the catalogs that a given user can access.
```

```
OLEDB::CONSTRAINT_COLUMN_USAGE(<"Catalog", "Schema", "Table-Name", "Column-Name")
returns the columns that referential constraints, unique constraints, check constraints, and assertions use that are defined in the catalog and that a given user owns.
```
OLEDB::CONSTRAINT_TABLE_USAGE("Catalog", "Schema", "Table-Name")
returns the tables that referential constraints, unique constraints, check constraints, and assertions use that are defined in the catalog and that a given user owns.

OLEDB::FOREIGN_KEYS("Primary-Key-Catalog", "Primary-Key-Schema", "Primary-Key-Table-Name", "Foreign-Key-Catalog", "Foreign-Key-Schema", "Foreign-Key-Table-Name")
returns the foreign key columns that a given user defined in the catalog.

OLEDB::INDEXES("Catalog", "Schema", "Index-Name", "Type", "Table-Name")
returns the indexes that are defined in the catalog that a given user owns.

OLEDB::KEY_COLUMN_USAGE("Constraint-Catalog", "Constraint-Schema", "Constraint-Name", "Table-Catalog", "Table-Schema", "Table-Name", "ColumnName")
returns the columns that are defined in the catalog and that a given user has constrained as keys.

OLEDB::PRIMARY_KEYS("Catalog", "Schema", "Table-Name")
returns the primary key columns that a given user defined in the catalog.

OLEDB::PROCEDURE_COLUMNS("Catalog", "Schema", "Procedure-Name", "Column-Name")
returns information about the columns of rowsets that procedures return.

OLEDB::PROCEDURE_PARAMETERS("Catalog", "Schema", "Procedure-Name", "Parameter-Name")
returns information about the parameters and return codes of the procedures.

OLEDB::PROCEDURES("Catalog", "Schema", "Procedure-Name", "Procedure-Type")
returns procedures that are defined in the catalog that a given user owns.

OLEDB::PROVIDER_INFO()
returns output that contains these columns: PROVIDER_NAME, PROVIDER_DESCRIPTION, and PROVIDER_PROPERTIES. The PROVIDER_PROPERTIES column contains a list of all properties that the provider supports. A semicolon (;) separates the properties. See “Examples of Special OLE DB Queries” on page 768.

OLEDB::PROVIDER_TYPES("Data Type", "Best-Match")
returns information about the base data types that the data provider supports.

OLEDB::REFERENTIAL_CONSTRAINTS("Catalog", "Schema", "Constraint-Name")
returns the referential constraints that are defined in the catalog that a given user owns.

OLEDB::SCHEMATA("Catalog", "Schema", "Owner")
returns the schemas that a given user owns.

OLEDB::SQL_LANGUAGES()
returns the conformance levels, options, and dialects that the SQL implementation processing data supports and that is defined in the catalog.

OLEDB::STATISTICS("Catalog", "Schema", "Table-Name")
returns the statistics that is defined in the catalog that a given user owns.

OLEDB::TABLE_CONSTRAINTS("Constraint-Catalog", "Constraint-Schema", "Constraint-Name", "Table-Catalog", "Table-Schema", "Table-Name", "Constraint-Type")
returns the table constraints that is defined in the catalog that a given user owns.
OLEDB::TABLE_PRIVILEGES(<"Catalog", "Schema", "Table-Name", "Grantor", "Grantee">)
returns the privileges on tables that are defined in the catalog that a given user grants or can access.

OLEDB::TABLES(<"Catalog", "Schema", "Table-Name", "Table-Type">)
returns the tables defined in the catalog that a given user grants and can access.

OLEDB::TRANSLATIONS(<"Catalog", "Schema", "Translation-Name">)
returns the character translations that are defined in the catalog and that are accessible to a given user.

OLEDB::USAGE_PRIVILEGES(<"Catalog", "Schema", "Object-Name", "ObjectType", "Grantor", "Grantee">)
returns the USAGE privileges on objects that are defined in the catalog and that a given user grants.

OLEDB::VIEW_COLUMN_USAGE(<"Catalog", "Schema", "View-Name">)
returns the columns on which viewed tables depend that are defined in the catalog and that a given user owns.

OLEDB::VIEW_TABLE_USAGE(<"Catalog", "Schema", "View-Name">)
returns the tables on which viewed tables depend that are defined in the catalog and that a given user owns.

OLEDB::VIEWS(<"Catalog", "Schema", "Table-Name">)
returns the viewed tables that are defined in the catalog and that a given user can access.

For a complete description of each rowset and the columns that are defined in each rowset, see the Microsoft OLE DB Programmer’s Reference.

**Examples of Special OLE DB Queries**

This example retrieves a rowset that displays all tables that the HRDEPT schema accesses:

```sql
proc sql;
    connect to oledb(provider=sqloledb properties={"User ID"=myusr1
                                        "Data Source"='dwtsrv1'});
    select * from oledb
    (OLEDB::TABLES("HRDEPT"));
quit;
```

It uses the special query OLEDB::PROVIDER_INFO() to produce this output:

```sql
proc sql;
    connect to oledb(provider=msdaora properties={"User ID"=myusr1
                                        "Data Source"="Oraserver"});
    select * from oledb
    (OLEDB::PROVIDER_INFO());
quit;
```
Output 27.1  Provider and Properties Output

<table>
<thead>
<tr>
<th>PROVIDER_NAME</th>
<th>PROVIDER_DESCRIPTION</th>
<th>PROVIDER_PROPERTIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSDAORA</td>
<td>Microsoft OLE DB Provider for Oracle</td>
<td>Password;User ID;Data Source;Window Handle;Locale Identifier;OLE DB Services; Prompt; Extended Properties;</td>
</tr>
<tr>
<td>SampProv</td>
<td>Microsoft OLE DB Sample Provider</td>
<td>Data Source;Window Handle; Prompt;</td>
</tr>
</tbody>
</table>

You could then reference the output when automating a connection to the provider. For the previous result set, you could write this SAS/ACCESS LIBNAME statement:

```sas
libname mydblib oledb provider=msdaora
  props=('Data Source'=OraServer 'User ID'=myusr1 'Password'=mypwd1);
```

Temporary Table Support for OLE DB

SAS/ACCESS Interface to OLE DB supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

Passing SAS Functions to OLE DB

SAS/ACCESS Interface to OLE DB passes the following SAS functions for OLE DB to DB2, Microsoft SQL Server, and Oracle for processing. Where the OLE DB function name differs from the SAS function name, the OLE DB name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

ABS  MAX
ARCOS (ACOS)  MIN
ARSIN (ASIN)  MOD (see note)
ATAN  QRT
AVG  REPEAT
BYTE  SIGN
CEIL  SIN
COS  SOUNDEX
COUNT  SQRT
DATE  STRIP (TRIM)
DATEPART  SUBSTR
DATETIME  SUM
EXP  TAN
FLOOR  TIME
INDEX  TIMEPART
LENGTH  TODAY
Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to OLE DB. Due to incompatibility in date and time functions between OLE DB and SAS, OLE DB might not process them correctly. Check your results to determine whether these functions are working as expected.

<table>
<thead>
<tr>
<th>Function</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESS</td>
<td>MONTH</td>
</tr>
<tr>
<td>DAY</td>
<td>SECOND</td>
</tr>
<tr>
<td>HOUR</td>
<td>WEEKDAY</td>
</tr>
<tr>
<td>MINUTE</td>
<td>YEAR</td>
</tr>
</tbody>
</table>

---

### Passing Joins to OLE DB

For a multiple libref join to pass to OLE DB, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- data source (DATASOURCE=)
- provider (PROVIDER=)
- qualifier (QUALIFIER=, if specified)
- provider string (PROVIDER_STRING, if specified)
- path and filename (UDL_FILE=, if specified)
- initialization string (INIT_STRING=, if specified)
- read isolation level (READ_ISOLATION_LEVEL=, if specified)
- update isolation level (UPDATE_ISOLATION_LEVEL=, if specified)
- all properties (PROPERTIES=)
- prompt (PROMPT=, must not be specified)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.
Bulk Loading for OLE DB

The BULKLOAD= LIBNAME option calls the SQLOLEDB interface of IRowsetFastLoad so that you can efficiently insert rows of data into a Microsoft SQL Server database table as a unit. BCP= is an alias for this option.

Note: This functionality is available only when accessing Microsoft SQL Server data on Windows platforms using Microsoft SQL Server Version 7.0 or later.

As the OLE DB interface sends rows of data to the bulk-load facility, data is written to an input buffer. When you send all rows or when the buffer reaches a certain size (DBCOMMIT= determines this), all rows are inserted as a unit into the table and the data is committed to the table. You can also set DBCOMMIT= to commit rows after a specified number of rows are inserted.

If an error occurs, a message is written to the SAS log, and any rows that were inserted before the error are rolled back.

If you specify BULKLOAD=YES and the PROVIDER= option is set, SAS/ACCESS Interface to OLE DB uses the specified provider. If you specify BULKLOAD=YES and PROVIDER= is not set, the engine uses the PROVIDER=SQLOLEDB value.

If you specify BULKLOAD=YES, connections that are made through OLE DB Services or UDL files are not allowed.

Locking in the OLE DB Interface

These LIBNAME and data set options let you control how the OLE DB interface handles locking. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

- READ_LOCK_TYPE= ROW | NOLOCK
- UPDATE_LOCK_TYPE= ROW | NOLOCK
- READ_ISOLATION_LEVEL= S | RR | RC | RU

The data provider sets the default value. OLE DB supports the S, RR, RC, and RU isolation levels that are defined in this table.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (serializable)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads.</td>
</tr>
<tr>
<td>RR (repeatable Read)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RC (committed Read )</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
</tbody>
</table>
Isolation Level | Definition
--- | ---
RU (uncommitted Read) | Allows dirty Reads, nonrepeatable Reads, and phantom Reads.

Here is how the terms in the table are defined.

**Dirty reads**
A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it can see changes that are made by those concurrent transactions even before they commit.

For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.

**Nonrepeatable reads**
If a transaction exhibits this phenomenon, it might read a row once. Then, if the same transaction attempts to read that row again, the row might have been changed or even deleted by another concurrent transaction. Therefore, the Read is not (necessarily) repeatable.

For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.

**Phantom reads**
When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist, a phantom.

- `UPDATE_ISOLATION_LEVEL = S | RR | RC`

The default value is set by the data provider. OLE DB supports the S, RR, and RC isolation levels defined in the preceding table. The RU isolation level is not allowed with this option.

## Accessing OLE DB for OLAP Data

### Overview

SAS/ACCESS Interface to OLE DB provides a facility for accessing OLE DB for OLAP data. You can specify a Multidimensional Expressions (MDX) statement through the SQL pass-through facility to access the data directly, or you can create an SQL view of the data. If your MDX statement specifies a data set with more than five axes (COLUMNS, ROWS, PAGES, SECTIONS, and CHAPTERS), SAS returns an error. See the Microsoft Data Access Components Software Developer's Kit for details about MDX syntax.
Note: This implementation provides Read-Only access to OLE DB for OLAP data. You cannot update or insert data with this facility.

Using the SQL Pass-Through Facility with OLAP Data

Overview
The main difference between normal OLE DB access using the SQL pass-through facility and the implementation for OLE DB for OLAP is the use of these additional identifiers to pass MDX statements to the OLE DB for OLAP data:

- **MDX::** identifies MDX statements that return a flattened data set from the multidimensional data.
- **MDX_DESCRIBE::** identifies MDX statements that return detailed column information.

An MDX_DESCRIBE:: identifier is used to obtain detailed information about each returned column. During the process of flattening multidimensional data, OLE DB for OLAP builds column names from each level of the given dimension. For example, for OLE DB for OLAP multidimensional data that contains CONTINENT, COUNTRY, REGION, and CITY dimensions, you could build a column with this name:

```
[NORTH AMERICA].[USA].[SOUTHEAST].ATLANTA
```

This name cannot be used as a SAS variable name because it has more than 32 characters. For this reason, the SAS/ACCESS engine for OLE DB creates a column name based on a shortened description, in this case, ATLANTA. However, because there could be an ATLANTA in some other combination of dimensions, you might need to know the complete OLE DB for OLAP column name. Using the MDX_DESCRIBE:: identifier returns a SAS data set that contains the SAS name for the returned column and its corresponding OLE DB for OLAP column name:

<table>
<thead>
<tr>
<th>SASNAME</th>
<th>MDX_UNIQUE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLANTA</td>
<td>[NORTH AMERICA].[USA].[SOUTHEAST].ATLANTA</td>
</tr>
<tr>
<td>CHARLOTTE</td>
<td>[NORTH AMERICA].[USA].[SOUTHEAST].[CHARLOTTE]</td>
</tr>
</tbody>
</table>

If two or more SASNAME values are identical, a number is appended to the end of the second and later instances of the name. For example, the values might be called ATLANTA, ATLANTA0, ATLANTA1, and so on. Also, depending on the value of the VALIDVARNAME= system option, invalid characters are converted to underscores in the SASNAME value.

Syntax
This facility uses the following general syntax. For more information about SQL pass-through facility syntax, see “SQL Pass-Through Facility” on page 471.
PROC SQL <options>;
  CONNECT TO OLEDB (<options>);
  <non-SELECT SQL statement(s)>
  SELECT column-identifier(s) FROM CONNECTION TO OLEDB
    ( MDX:: | MDX_DESCRIBE:: <MDX statement> )
  <other SQL statement(s)>
;

Examples
This code uses the SQL pass-through facility to pass an MDX statement to a Microsoft SQL Server Decision Support Services (DSS) Cube. The provider used is the Microsoft OLE DB for OLAP provider named MSOLAP.

proc sql noerrorstop;
  connect to oledb (provider=msolap prompt=yes);
  select * from connection to oledb
    ( MDX::select {[Measures].[Units Shipped],
      [Measures].[Units Ordered]} on columns,
      NON EMPTY [Store].[Store Name].members on rows
      from Warehouse );

See the Microsoft Data Access Components Software Developer's Kit for details about MDX syntax.

The CONNECT statement requests prompting for connection information, which facilitates the connection process (especially with provider properties). The MDX:: prefix identifies the statement within the parentheses that follows the MDX statement syntax, and is not an SQL statement that is specific to OLAP. Partial output from this query might look like this:

<table>
<thead>
<tr>
<th>Store</th>
<th>Units Shipped</th>
<th>Units Ordered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store6</td>
<td>10,647</td>
<td>11,699</td>
</tr>
<tr>
<td>Store7</td>
<td>24,850</td>
<td>26,223</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

You can use the same MDX statement with the MDX_DESCRIBE:: identifier to see the full description of each column:

proc sql noerrorstop;
  connect to oledb (provider=msolap prompt=yes);
  select * from connection to oledb
    ( MDX_DESCRIBE::select {[Measures].[Units Shipped],
      [Measures].[Units Ordered]} on columns,
      NON EMPTY [Store].[Store Name].members on rows
      from Warehouse );

The next example creates a view of the OLAP data, which is then accessed using the PRINT procedure:

proc sql noerrorstop;
  connect to oledb(provider=msolap
    props=('data source'=sqlserverdb
      'user id'=myuserid password=mypassword));
  create view work.myview as
    select * from connection to oledb
In this example, full connection information is provided in the CONNECT statement, so the user is not prompted. The SQL view can be used in other PROC SQL statements, the DATA step, or in other procedures. However, you cannot modify (that is, insert, update, or delete a row in) the view's underlying multidimensional data.

---

**Naming Conventions for OLE DB**

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Because OLE DB is an application programming interface (API), data source names for files, tables, and columns are determined at run time. Most SAS names can be up to 32 characters long. SAS/ACCESS Interface to OLE DB also supports file, table, and column names up to 32 characters long. If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a name results in identical names, SAS generates unique names by replacing the last character with a number. For more information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.)

---

**Data Types for OLE DB**

**Overview**

Each data source column in a table has a name and a data type. The data type tells the data source how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about OLE DB null and default values and data conversions.

**OLE DB Null Values**

Many relational database management systems have a special value called NULL. A DBMS NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DBMS NULL value, it interprets it as a SAS missing value.

In most relational databases, columns can be defined as NOT NULL so that they require data (they cannot contain NULL values). When a column is defined as NOT NULL, the
DBMS does not add a row to the table unless the row has a value for that column. When creating a DBMS table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

OLE DB mirrors the behavior of the underlying DBMS with regard to NULL values. See the documentation for your DBMS for information about how it handles NULL values.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how the DBMS handles SAS missing character values, use the NULLCHAR= and NULLCHARVAL= data set options.

Setting the BOOL_VAL Environment Variable

By default, the SAS/ACCESS LIBNAME engine imports YES (TRUE) into SAS as the numeric value 1. Set the environment variable with this statement.

/* To have the YES value imported into SAS as numeric value-1 */
OPTIONS SET=BOOL_VAL ASIS;

/* Reset to the default value */
OPTIONS SET=BOOL_VAL SAS;

Note: Support for the BOOL_VAL environment variable was added for SAS 9.4.

LIBNAME Statement Data Conversions

This table shows all data types and default SAS formats that SAS/ACCESS Interface to OLE DB supports. It does not explicitly define the data types as they exist for each data source. It lists the types that each data source's data type might map to. For example, an INTEGER data type under DB2 might map to an OLE DB data type of DBTYPE_I4. All data types are supported.

Table 27.4  OLE DB Data Types and Default SAS Formats

<table>
<thead>
<tr>
<th>OLE DB Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTYPE_R8</td>
<td>none</td>
</tr>
<tr>
<td>DBTYPE_R4</td>
<td>none</td>
</tr>
<tr>
<td>DBTYPE_I8</td>
<td>none</td>
</tr>
<tr>
<td>DBTYPE_UI8</td>
<td>none</td>
</tr>
<tr>
<td>DBTYPE_I4</td>
<td>11.</td>
</tr>
<tr>
<td>DBTYPE_UI4</td>
<td>11.</td>
</tr>
<tr>
<td>DBTYPE_I2</td>
<td>6.</td>
</tr>
<tr>
<td>DBTYPE_UI2</td>
<td>6.</td>
</tr>
</tbody>
</table>
The following table shows the default data types that SAS/ACCESS Interface to OLE DB uses when creating DBMS tables. SAS/ACCESS Interface to OLE DB lets you specify non-default data types by using the DBTYPE= data set option.

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Default OLE DB Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.$</td>
<td>DBTYPE_STR using $n^*$</td>
</tr>
<tr>
<td>date formats</td>
<td>DBTYPE_DBDATE</td>
</tr>
<tr>
<td>time formats</td>
<td>DBTYPE_DBTIME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OLE DB Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTYPE_I1</td>
<td>4.</td>
</tr>
<tr>
<td>DBTYPE_UI1</td>
<td>4.</td>
</tr>
<tr>
<td>DBTYPE_BOOL</td>
<td>1.</td>
</tr>
<tr>
<td>DBTYPE_NUMERIC</td>
<td>$m$ or $w.d$ or none, if $w$ and $d$ are not specified</td>
</tr>
<tr>
<td>DBTYPE_DECIMAL</td>
<td>$w$ or $w.d$ or none, if $w$ and $d$ are not specified</td>
</tr>
<tr>
<td>DBTYPE_CY</td>
<td>DOLLAR$w.2$</td>
</tr>
<tr>
<td>DBTYPE_BYTES</td>
<td>$w.$</td>
</tr>
<tr>
<td>DBTYPE_STR</td>
<td>$w.$</td>
</tr>
<tr>
<td>DBTYPE_BSTR</td>
<td>$w.$</td>
</tr>
<tr>
<td>DBTYPE_WSTR</td>
<td>$w.$</td>
</tr>
<tr>
<td>DBTYPE_VARIANT</td>
<td>$w.$</td>
</tr>
<tr>
<td>DBTYPE_DBDATE</td>
<td>DATE9.</td>
</tr>
<tr>
<td>DBTYPE_DBTIME</td>
<td>TIME8.</td>
</tr>
<tr>
<td>DBTYPE_DBTIMESTAMP</td>
<td>DATETIME$w.d$, where $w$ depends on precision and $d$ depends on scale</td>
</tr>
<tr>
<td>DBTYPE_DATE</td>
<td></td>
</tr>
<tr>
<td>DBTYPE_GUID</td>
<td>$38.$</td>
</tr>
<tr>
<td>SAS Variable Format</td>
<td>Default OLE DB Data Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>datetime formats</td>
<td>DBTYPE_DBTIMESTAMP</td>
</tr>
</tbody>
</table>

* n in OLE DB character data types is equivalent to w in SAS formats.

** m and n in OLE DB numeric data types are equivalent to w and d in SAS formats.
Chapter 28
SAS/ACCESS Interface to Oracle

Introduction to SAS/ACCESS Interface to Oracle ........................................ 780
LIBNAME Statement Specifics for Oracle .................................................. 780
  Overview .................................................................................................. 780
  Arguments ............................................................................................... 780
  Oracle LIBNAME Statement Examples .................................................... 784
Data Set Options for Oracle ......................................................................... 784
SQL Pass-Through Facility Specifics for Oracle .......................................... 787
  Key Information ...................................................................................... 787
  CONNECT Statement Examples ............................................................. 788
Autopartitioning Scheme for Oracle ............................................................ 789
  Overview .................................................................................................. 789
  Partitioned Oracle Tables ...................................................................... 790
  Nonpartitioned Oracle Tables ................................................................. 791
  Performance Summary .......................................................................... 792
ACCESS Procedure Specifics for Oracle ..................................................... 792
  Overview .................................................................................................. 792
DBLOAD Procedure Specifics for Oracle ................................................... 794
  Overview .................................................................................................. 794
Maximizing Oracle Performance ............................................................... 796
Passing SAS Functions to Oracle ............................................................... 796
Passing Joins to Oracle ............................................................................... 797
Bulk Loading for Oracle ............................................................................ 798
  Overview .................................................................................................. 798
  Data Set Options with Bulk Loading ...................................................... 798
  Interactions with Other Options ............................................................ 799
  z/OS Specifics ......................................................................................... 799
  Example .................................................................................................. 800
Locking in the Oracle Interface ................................................................. 800
Naming Conventions for Oracle ................................................................. 801
Data Types for Oracle ................................................................................ 802
  Overview .................................................................................................. 802
  Supported Oracle Data Types ............................................................... 802
  Example 1: Timestamp ......................................................................... 803
  Example 2: Interval Year to Month ....................................................... 805
Introduction to SAS/ACCESS Interface to Oracle

For available SAS/ACCESS features, see Oracle supported features on page 86. For more information about Oracle, see your Oracle documentation.

LIBNAME Statement Specifics for Oracle

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Oracle supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Oracle.

```
LIBNAME libref oracle <connection-options> <LIBNAME-options>;
```

Arguments

```
libref
```

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

```
oracle
```

specifies the SAS/ACCESS engine name for the Oracle interface.

```
connection-options
```

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. Here is how these options are defined.

```
USER='Oracle-user-name'
```

specifies an optional Oracle user name. If the user name contains blanks or national characters, enclose it in quotation marks. If you omit an Oracle user name and password, the default Oracle user ID OP$sysid is used, if it is enabled. USER= must be used with PASSWORD=.

Restriction: When you specify `<Oracle-user-name>` in the LIBNAME statement, SAS changes the user name to uppercase characters, even if the user name uses quotation marks. Mixed case characters are not supported.
PASSWORD=’Oracle-password’
specifies an optional Oracle password that is associated with the Oracle user name. If you omit it, the password for the default Oracle user ID OPS$sysid is used, if it is enabled. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you specify USER=, you must specify PASSWORD=.

BUFFSIZE=number-of-rows
specifies the number of rows to retrieve from an Oracle table or view with each fetch. Using this argument can improve the performance of any query to Oracle.

By setting the value of the BUFFSIZE= argument in your SAS programs, you can find the optimal number of rows for a given query on a given table. The default buffer size is 250 rows per fetch. The value can be up to 2,147,483,647 rows per fetch, although a practical limit for most applications is less, depending on the available memory.

PATH=’Oracle-database-specification’
specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS. SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the databases that have been set up in your operating environment, and to determine the default values if you do not specify a database. On UNIX systems, the TWO_TASK environment variable is used, if set. If neither the PATH= nor the TWO_TASK values have been set, the default value is the local driver.

If you specify the appropriate system options or environment variables for Oracle, you can often omit the connection options from your LIBNAME statements. See your Oracle documentation for details.

LIBNAME-options
define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Oracle, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

Table 28.1 SAS/ACCESS LIBNAME Options for Oracle

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS=</td>
<td>conditional</td>
</tr>
<tr>
<td>ADJUST_NCHAR_COLUMN_LENGTHS=</td>
<td>YES</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DEFAULT_DIR=</td>
<td>&lt;database-name&gt;</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>DB_LENGTH_SEMANTICS_BYTE=</td>
<td>YES</td>
</tr>
<tr>
<td>DB_OBJECTS=</td>
<td>(TABLES VIEWS)</td>
</tr>
<tr>
<td>DBCLIENT_ENCODING_FIXED=</td>
<td>YES or NO, based on the SAS encoding</td>
</tr>
<tr>
<td>DBCLIENT_MAX_BYTES=</td>
<td>matches the maximum number of bytes per single character of the SAS session encoding</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating, deleting, or appending to an existing table)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DLBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DLBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBLINK=</td>
<td>the local database</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSERVER_ENCODING_FIXED=</td>
<td>YES or NO, based on the Oracle server encoding</td>
</tr>
<tr>
<td>DBSERVER_MAX_BYTES=</td>
<td>usually 1</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT ExE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>1 is the forced default when REREAD_EXPOSURE=YES; otherwise, 10</td>
</tr>
<tr>
<td>LOCKWAIT=</td>
<td>YES</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>NONE</td>
</tr>
<tr>
<td>OR_BINARY_DOUBLE=</td>
<td>YES</td>
</tr>
<tr>
<td>OR_ENABLE_INTERRUPT=</td>
<td>NO</td>
</tr>
<tr>
<td>OR_UPD_NOWHERE=</td>
<td>YES</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>NO</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>NO</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>250</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>see “Locking in the Oracle Interface”</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>NOLOCK</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>SAS accesses objects in the default and public schemas</td>
</tr>
<tr>
<td>SHOW_SYNONYMS=</td>
<td>YES</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SQLGENERATION=</td>
<td>DBMS</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>see “Locking in the Oracle Interface”</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>NOLOCK</td>
</tr>
<tr>
<td>UPDATEBUFF=</td>
<td>1</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>
**Oracle LIBNAME Statement Examples**

This example uses default settings for the connection options to make the connection. If you specify the appropriate system options or environment variables for Oracle, you can often omit the connection options from your LIBNAME statements. See your Oracle documentation for details.

```sas
libname myoralib oracle;
```

In this example, the MYDBLIB libref uses the Oracle interface to connect to an Oracle database. SAS/ACCESS connection options are USER=, PASSWORD=, and PATH=. PATH= specifies an alias for the database specification, which SQL*Net requires.

```sas
libname mydblib oracle user=myusr1 password=mypwd1 path=mysrv1;
```

```sas
proc print data=mydblib.employees;
  where dept='CSR010';
run;
```

This example connects to Oracle without updating the TNSNAMES.ORA file.

```sas
libname x oracle user=myusr1 pw=mypwd1 path="(DESCRIPTION=
ADDRESS_LIST=
  (ADDRESS= (PROTOCOL=TCP)(HOST=pinkfloyd)(PORT=1521))
  (CONNECT_DATA=
    (SID=alien)
    )
  )";
```

---

**Data Set Options for Oracle**

All SAS/ACCESS data set options in this table are supported for Oracle. Default values are provided where applicable. For details, see [Data Set Options for Relational Databases](#) on page 241.

**Table 28.2 SAS/ACCESS Data Set Options for Oracle**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_BADFILE=</td>
<td>creates a file in the current directory or with the default file specifications</td>
</tr>
<tr>
<td>BL_API_BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_CONTROL=</td>
<td>creates a file in the current directory or with the default file specifications</td>
</tr>
<tr>
<td>BL_DATAFILE=</td>
<td>creates a file in the current directory or with the default file specifications</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BL_DEFAULT_DIR=</td>
<td><code>&lt;database-name&gt;</code></td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_DELETE_ONLY_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DIRECT_PATH=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_DISCARDFILE=</td>
<td>creates a file in the current directory or with the default file specifications</td>
</tr>
<tr>
<td>BL_INDEX_OPTIONS=</td>
<td>the current SQL*Loader Index options with bulk-loading</td>
</tr>
<tr>
<td>BL_LOAD_METHOD=</td>
<td>When loading an empty table, the default value is INSERT. When loading a table that contains data, the default value is APPEND.</td>
</tr>
<tr>
<td>BL_LOG=</td>
<td>If a log file does not already exist, it is created in the current directory or with the default file specifications. If a log file does already exist, the Oracle bulk loader reuses the file, replacing the contents with information from the new load.</td>
</tr>
<tr>
<td>BL_OPTIONS=</td>
<td>ERRORS=1000000</td>
</tr>
<tr>
<td>BL_PRESERVE_BLANKS=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_RECOVERABLE=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_RETURN_WARNINGS_AS_ERRORS=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_SQLLDR_PATH=</td>
<td>sqldr</td>
</tr>
<tr>
<td>BL_SUPPRESS_NULLIF=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_USE_PIPE=</td>
<td>NO</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DB_ONE_CONNECT_PER THREAD=</td>
<td>YES</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBLINK=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for Oracle”</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “LIBNAME Statement Data Conversions”</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>OR_IDENTITY_COLS=</td>
<td>none</td>
</tr>
<tr>
<td>OR_PARTITION=</td>
<td>an Oracle table partition name</td>
</tr>
<tr>
<td>OR_UPD_NOWHERE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>ORHINTS=</td>
<td>no hints</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for Oracle

#### Key Information

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Oracle interface.

- The `dbms-name` is **oracle**.

- The CONNECT statement is optional. If you omit it, an implicit connection is made with your OPS$sysid, if it is enabled. When you omit a CONNECT statement, an implicit connection is performed when the first EXECUTE statement or CONNECTION TO component is passed to Oracle. In this case you must use the default DBMS name **oracle**.

- The Oracle interface can connect to multiple databases (both local and remote) and to multiple user IDs. If you use multiple simultaneous connections, you must use an alias argument to identify each connection. If you do not specify an alias, the default alias, **oracle**, is used.

- Here are the `database-connection-arguments` for the CONNECT statement.

  ```
  USER=<>Oracle-user-name<> 
  ```

  specifies an optional Oracle user name. If you specify USER=, you must also specify PASSWORD=.
PASSWORD= '<Oracle-password>'
specifies an optional Oracle password that is associated with the Oracle user name. If you omit an Oracle password, the default Oracle user ID OPS$sysid is used, if it is enabled. If you specify PASSWORD=, you must also specify USER=.

Alias: ORAPW=

Tip: If you do not wish to enter your Oracle password in uncoded text, see PROC PWENCODE in the Base SAS Procedures Guide for a method to encode it.

BUFFSIZE=number-of-rows
specifies the number of rows to retrieve from an Oracle table or view with each fetch. Using this argument can improve the performance of any query to Oracle.

By setting the value of the BUFFSIZE= argument in your SAS programs, you can find the optimal number of rows for a given query on a given table. The default buffer size is 250 rows per fetch. The value can be up to 2,147,483,647 rows per fetch, although a practical limit for most applications is less, depending on the available memory.

PRESERVE_COMMENTS
lets you pass additional information (called hints) to Oracle for processing. These hints might direct the Oracle query optimizer to choose the best processing method based on your hint.

You specify PRESERVE_COMMENTS as an argument in the CONNECT statement. You then specify the hints in the Oracle SQL query for the CONNECTION TO component. Hints are entered as comments in the SQL query and are passed to and processed by Oracle.

PATH= '<Oracle-database-specification>'
specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS.

SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the path designations that have been set up in your operating environment and the default value if you do not specify a path designation. On UNIX systems, the TWO_TASK environment variable is used, if set. If neither PATH= nor TWO_TASK have been set, the default value is the local driver.

**CONNECT Statement Examples**

This example uses the alias DBCON for the DBMS connection. The connection alias is optional.

```sas
proc sql;
   connect to oracle as dbcon
   (user=myusr1 password=mypwd1 buffsize=100
    path='mysrv1');
quit;
```

This next example connects to Oracle and sends it two EXECUTE statements to process.

```sas
proc sql;
   connect to oracle (user=myusr1 password=mypwd1);
   execute (create view whotookorders as
```
select ordernum, takenby,
    firstname, lastname, phone
from orders, employees
where orders.takenby=employees.empid)
by oracle;
execute (grant select on whotookorders
to myusr1) by oracle;
disconnect from oracle;
quit;

As shown in highlighted text, this example performs a query on the CUSTOMERS Oracle table:

proc sql;
connect to oracle (user=myusr1 password=mypwd1);
select *
    from connection to oracle
       (select * from customers
          where customer like '1%');
disconnect from oracle;
quit;

In this example, the PRESERVE_COMMENTS argument is specified after the USER= and PASSWORD= arguments. The Oracle SQL query is enclosed in the required parentheses. The SQL INDX command identifies the index for the Oracle query optimizer to use to process the query. Multiple hints are separated with blanks.

proc sql;
connect to oracle as mycon(user=myusr1
    password=mypwd1 preserve_comments);
select *
    from connection to mycon
       (select /* +indx(empid) all_rows */
          count(*) from employees);
quit;

Hints are not preserved in this next example, which uses the prior style of syntax:

execute ( delete /*+ FIRST_ROWS */ from test2 where num2=1)
by &db

Using the new syntax, hints are preserved in this example:

execute by &db
   ( delete /*+ FIRST_ROWS */ from test2 where num2=2);

Autopartitioning Scheme for Oracle

Overview

Without user-specified partitioning from the DBSLICE= option, SAS/ACCESS Interface to Oracle tries to use its own partitioning techniques. The technique that it chooses depends on whether the table is physically partitioned on the Oracle server.

For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.
Threaded Reads for the Oracle engine on z/OS are not supported.

### Partitioned Oracle Tables

If you are working with a partitioned Oracle table, it is recommended that you let the Oracle engine partition the table for you. The Oracle engine gathers all partition information needed to perform a threaded Read on the table.

A partitioned Oracle table is a good candidate for a threaded Read because each partition in the table can be read in parallel with little contention for disk resources. If the Oracle engine determines that the table is partitioned, it makes the same number of connections to the server as there are partitions. This is true as long as the maximum number of threads that are allowed is higher than the number of partitions. Each connection retrieves rows from a single partition.

If the value of the maximum number of allowed threads is less than the number of partitions on the table, a single connection reads multiple partitions. Each connection retrieves rows from a single partition or multiple partitions. However, you can use the `DB_ONE_CONNECT_PER_THREAD=` data set option so that there is only one connection per thread.

The following example shows how to do this. First, create the SALES table in Oracle.

```sql
CREATE TABLE SALES (acct_no NUMBER(5),
  acct_name CHAR(30), amount_of_sale NUMBER(6), qtr_no INTEGER)
PARTITION BY RANGE (qtr_no)
(PARTITION sales1 VALUES LESS THAN (2) TABLESPACE ts0,
 PARTITION sales2 VALUES LESS THAN (2) TABLESPACE ts1,
 PARTITION sales3 VALUES LESS THAN (2) TABLESPACE ts2,
 PARTITION sales4 VALUES LESS THAN (2) TABLESPACE ts3)
```

Performing a threaded Read on this table with the following code, SAS makes four separate connections to the Oracle server and each connection reads from each partition. Turning on `SASTRACE=` shows the SQL that is generated for each connection.

```
libname x oracle user=myusr1 path=mysrv1;
data new;
set x.SALES (DBSLICEPARM=(ALL,10));
run;
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES2)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES3)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES1)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES4)
```

Using the following code, SAS instead makes two separate connections to the Oracle server and each connection reads from two different partitions.

```
libname x oracle user=myusr1 path=mysrv1;
data new;
set x.SALES (DBSLICEPARM=(ALL,2));
run;
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES2) UNION ALL SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES3)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES4)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES1)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES2)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES3)
```

```
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES partition (SALES4)
```
Using DB_ONE_CONNECT_PER_THREAD=NO, however, you can override the default behavior of limiting the number of connections to the number of threads. As shown below, SAS makes four separate connections to the Oracle server and each connection reads from each of the partition.

libname x oracle user=myusr1 path=mysrv1;
data new;
set x.SALES (DBSLICEPARM=(ALL,2) DB_ONE_CONNECT_PER_THREAD=NO );
run;

The second parameter of the DBSLICEPARM= LIBNAME option determines the number of threads to read the table in parallel. The number of partitions on the table, the maximum number of allowed threads, and the value of DB_ONE_CONNECT_PER_THREAD determine the number of connections to the Oracle server for retrieving rows from the table.

**Nonpartitioned Oracle Tables**

If the table is not partitioned and the DBSLICE= option is not specified, Oracle resorts to the MOD function. (See "Autopartitioning Techniques in SAS/ACCESS" on page 59.) With this technique, the engine makes \( N \) connections, and each connection retrieves rows based on a WHERE clause as follows:

\[
\text{WHERE } \text{ABS} \left( \text{MOD(} \text{ModColumn} , N \right) ) = R
\]

- \( \text{ModColumn} \) is a column in the table of type integer and is not used in any user specified WHERE clauses. (The engine selects this column. If you do not think this is the ideal partitioning column, you can use the DBSLICE= data set option to override this default behavior.)
- \( R \) varies from 0 to \( (N-1) \) for each of the \( N \) WHERE clauses.
- \( N \) defaults to 2, and \( N \) can be overridden with the second parameter in the DBSLICEPARM= data set option.

The Oracle engine selects the ModColumn to use in this technique. Any numeric column with zero scale value can qualify as the ModColumn. However, if a primary key column is present, it is preferred over all others. Generally, values in the primary key column are in a serial order and yield an equal number of rows for each connection. This example illustrates the point:

```
create table employee (empno number(10) primary key,  
    empname varchar2(20), hiredate date,  
    salary number(8,2), gender char(1));
```
Performing a threaded Read on this table causes Oracle to make two separate connections to the Oracle server. SAS tracing shows the SQL generated for each connection:

data new;
set x.EMPLOYEE(DBSLICPARM=ALL);
run;

ORACLE: SELECT "EMPNO", "EMPNAME", "HIREDATE", "SALARY", "GENDER"
FROM EMPLOYEE WHERE ABS(MOD("EMPNO",2))=0

ORACLE: SELECT "EMPNO", "EMPNAME", "HIREDATE", "SALARY", "GENDER"
FROM EMPLOYEE WHERE ABS(MOD("EMPNO",2))=1

EMPNO, the primary key, is selected as the MOD column.

The success of MOD depends on the distribution of the values within the selected ModColumn and the value of N. Ideally, the rows are distributed evenly among the threads.

You can alter the N value by changing the second parameter of DBSLICEPARM= LIBNAME option.

**Performance Summary**

There are times that you might not see an improvement in performance with the MOD technique. It is possible that the engine might not be able to find a column that qualifies as a good MOD column. In these situations, you can explicitly specify DBSLICE= data set option to force a threaded Read and improve performance.

It is a good policy to let the engine autopartition and intervene with DBSLICE= only when necessary.

---

**ACCESS Procedure Specifics for Oracle**

**Overview**

For general information about this feature, see Appendix 1, “ACCESS Procedure,” on page 955. Oracle examples are available.

The Oracle interface supports all ACCESS procedure statements in line and batch modes. See “About ACCESS Procedure Statements” on page 956.

Here are the ACCESS procedure specifics for Oracle.

- The PROC ACCESS step DBMS= value is Oracle.
- Here are the database-description-statements that PROC ACCESS uses:

  - USER=<>Oracle-user-name<>
    specifies an optional Oracle user name. If you omit an Oracle password and user name, the default Oracle user ID OPSSysid is used if it is enabled. If you specify USER=, you must also specify ORAPW=.

  - ORAPW= <>Oracle-password<>
    specifies an optional Oracle password that is associated with the Oracle user name. If you omit ORAPW=, the password for the default Oracle user ID OPSSysid is used, if it is enabled. If you specify ORAPW=, you must also specify USER=.
PATH=

specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS.

SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the path designations that are set up in your operating environment and the default value if you do not specify a path designation. On UNIX systems, the TWO_TASK environment variable is used, if set. If neither PATH= nor TWO_TASK have been set, the default value is the local driver.

• Here is the PROC ACCESS step TABLE= statement:

    TABLE= <Oracle-table-name>;

specifies the name of the Oracle table or Oracle view on which the access descriptor is based. This statement is required. The Oracle-table-name argument can be up to 30 characters long and must be a valid Oracle table name. If the table name contains blanks or national characters, enclose it in quotation marks.

Examples

Example 1: Create an Access and a View Descriptor

This example creates an access descriptor and a view descriptor based on Oracle data.

    options linesize=80;

    libname adlib 'SAS-library';
    libname vlib 'SAS-library';

    proc access dbms=oracle;
    /* Create an access descriptor */
        create adlib.customer.access;
        user=myusr1;
        orapw=mypwd1;
        table=customers;
        path='mysrv1';
        assign=yes;
        rename customer=custnum;
        format firstorder date9.;
        list all;

    /* Create a view descriptor */
        create vlib.usacust.view;
        select customer state zipcode name firstorder;
        subset where customer like '1%';
    run;
Example 2: Create and Print a View Descriptor

This example creates another view descriptor that is based on the ADLIB.CUSTOMER access descriptor. You can then print the view.

```sas
/* Create the socust view */
proc access dbms=oracle accdesc=adlib.customer;
    create vlib.socust.view;
    select customer state name contact;
    subset where state in ('NC', 'VA', 'TX');
run;

/* Print the socust view */
proc print data=vlib.socust;
    title 'Customers in Southern States';
run;
```

DBLOAD Procedure Specifics for Oracle

Overview

For general information about this feature, see Appendix 3, “DBLOAD Procedure,” on page 985. Oracle examples are available.

The Oracle interface supports all DBLOAD procedure statements. See “About DBLOAD Procedure Statements” on page 986.

Here are the DBLOAD procedure specifics for Oracle.

- The PROC DBLOAD step DBMS= value is Oracle.
- Here are the database-description-statements that PROC DBLOAD uses:
  - USER=<Oracle-user-name> specifies an optional Oracle user name. If you omit an Oracle password and user name, the default Oracle user ID OPSSysid is used if it is enabled. If you specify USER=, you must also specify ORAPW=.
  - ORAPW=<Oracle-password> specifies an optional Oracle password that is associated with the Oracle user name. If you omit ORAPW=, the password for the default Oracle user ID OPSSysid is used, if it is enabled. If you specify ORAPW=, you must also specify USER=.
  - PATH=<Oracle-database-specification> specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS.

SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the path designations that are set up in your operating environment, and the default value if you do not specify a path designation. On UNIX systems, the TWO_TASK

794  Chapter 28  •  SAS/ACCESS Interface to Oracle
environment variable is used, if set. If neither PATH= nor TWO_TASK have
been set, the default value is the local driver.

TABLESPACE= '<Oracle-tablespace-name>';  
    specifies the name of the Oracle table space where you want to store the new
table. The Oracle-tablespace-name argument can be up to 18 characters long and
must be a valid Oracle table space name. If the name contains blanks or national
characters, enclose the entire name in quotation marks.

    If TABLESPACE= is omitted, the table is created in your default table space that
is defined by the Oracle database administrator at your site.

• Here is the PROC DBLOAD step TABLE= statement:

TABLE= '<Oracle-table-name>';  
    specifies the name of the Oracle table or Oracle view on which the access
descriptor is based. This statement is required. The Oracle-table-name argument
can be up to 30 characters long and must be a valid Oracle table name. If the
table name contains blanks or national characters, enclose the name in quotation
marks.

Examples

Example 1: Create an Oracle Table from a Data File

This example creates a new Oracle table, EXCHANGE, from the DLIB.RATEOFEX
data file. Based on the new table, an access descriptor, ADLIB.EXCHANGE, is also
created. The PATH= statement uses an alias to connect to a remote Oracle7 Server
database. The SQL statement in the second DBLOAD procedure sends an SQL GRANT
statement to Oracle. You must be granted Oracle privileges to create new Oracle tables
or to grant privileges to other users. The SQL statement is in a separate procedure
because you cannot create a DBMS table and reference it within the same DBLOAD
step. The new table is not created until the RUN statement is processed at the end of the
first DBLOAD step.

libname adlib 'SAS-library';
libname dlib 'SAS-library';

proc dbload dbms=oracle data=dlib.rateofex;
    user=myusr1;
    orapw=mypwd1;
    path='mysrv1';
    table=exchange;
    accdesc=adlib.exchange;
    rename fgnindol=fgnindolar 4=dolrsinfgn;
    nulls updated=n fgnindol=n 4=n country=n;
    load;
run;

proc dbload dbms=oracle;
    user=myusr1;
    orapw=mypwd1;

Example 2: Append Rows to a Table from a Data Set

This example uses the APPEND option to append rows from the INVDATA data set. This data set was created previously to an existing Oracle table named INVOICE.

```sql
proc dbload dbms=oracle data=invdata append;
user=myusr1;
orapw=mypwd1;
path='mysrv1';
table=invoice;
load;
run;
```

Maximizing Oracle Performance

You can take several measures to optimize performance when using SAS/ACCESS Interface to Oracle. For general information about improving performance when using SAS/ACCESS engines, see Chapter 4, “Performance Considerations,” on page 37.

SAS/ACCESS Interface to Oracle has several options that you can use to further improve performance.

- For tips on multi-row processing, see these LIBNAME options: INSERTBUFF, UPDATEBUFF, and READBUFF.
- For instructions on using the Oracle SQL*Loader to increase performance when loading rows of data into Oracle tables, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

If you choose the transactional inserting of rows (specify BULKLOAD=NO), you can improve performance by inserting multiple rows at a time. This performance enhancement is comparable to using the Oracle SQL*Loader Conventional Path Load. For more information about inserting multiple rows, see the INSERTBUFF= option.

Passing SAS Functions to Oracle

SAS/ACCESS Interface to Oracle passes the following SAS functions to Oracle for processing. Where the Oracle function name differs from the SAS function name, the Oracle name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

- ABS
- ARCOS (ACOS)
- ARSIN (ASIN)
- ATAN
- ATAN2
- AVG
- MAX
- MIN
- MINUTE
- SECOND
- SIGN
- SIN
When the Oracle server is 9i or above, these additional functions are also passed.

\[
\begin{align*}
\text{COALESCE} & \quad \text{MONTH (EXTRACT)} \\
\text{DAY (EXTRACT)} & \quad \text{YEAR (EXTRACT)} \\
\end{align*}
\]

\text{SQL\_FUNCTIONS=ALL} allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when \text{SQL\_FUNCTIONS=ALL} can the SAS/ACCESS engine also pass these SAS SQL functions to Oracle. Due to incompatibility in date and time functions between Oracle and SAS, Oracle might not process them correctly. Check your results to determine whether these functions are working as expected.

\[
\begin{align*}
\text{DATE (TRUNC(SYSDATE))} & \quad \text{SUBSTR} \\
\text{DATEPART (TRUNC)} & \quad \text{TODAY (TRUNC(SYSDATE))} \\
\text{INDEX (INSTR)} & \quad \text{TRANWRD (REPLACE)} \\
\text{LENGTH} & \quad \text{TRIM (TRIMN)} \\
\text{MOD} & \\
\end{align*}
\]

*Only in WHERE or HAVING clauses.

---

**Passing Joins to Oracle**

Before a join can pass to Oracle, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- path (PATH=)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.
Bulk Loading for Oracle

Overview

SAS/ACCESS Interface to Oracle can call the Oracle SQL*Loader (SQLLDR) when you set the option BULKLOAD=YES. The Oracle bulk loader provides superior load performance, so you can rapidly move data from a SAS file into an Oracle table. Future releases of SAS/ACCESS software will continue to use powerful Oracle tools to improve load performance.

Data Set Options with Bulk Loading

Here are the Oracle bulk-load data set options. For detailed information about these options, see Data Set Options for Relational Databases on page 241.

- BL_BADFILE=
- BL_CONTROL=
- BL_DATAFILE=
- BL_DEFAULT_DIR=
- BL_DELETE_DATAFILE=
- BL_DIRECT_PATH=
- BL_DISCARDFILE=
- BL_INDEX_OPTIONS=
- BL_LOAD_METHOD=
- BL_LOG=
- BL_OPTIONS=
- BL_PARFILE=
- BL_PREVENT_BLANKS=
- BL_RECOVERABLE=
- BL_RETURN_WARNINGS_AS_ERRORS=
- BL_SQLLDR_PATH=
- BL_SUPPRESS_NULLIF=
- BULKLOAD=

BULKLOAD= calls the Oracle bulk loader so that the Oracle engine can move data from a SAS file into an Oracle table using SQL*Loader (SQLLDR).

Note: SQL*Loader direct-path load has a number of limitations. See your Oracle utilities documentation for details, including tips to boost performance. You can also view the SQL*Loader log file instead of the SAS log for information about the load when you use bulk loading.
Interactions with Other Options

When BULKLOAD=YES, the following statements are true:

- The `DBCOMMIT=`, `DBFORCE=`, `ERRLIMIT=`, and `INSERTBUFF=` options are ignored.
- If `NULLCHAR=SAS`, and the `NULLCHARVAL=` value is blank, the SQL*Loader attempts to insert a NULL instead of a NULLCHARVAL value.
- If `NULLCHAR=NO`, and the `NULLCHARVAL` value is blank, the SQL*Loader attempts to insert a NULL even if the DBMS does not allow NULL.

To avoid this result, set `BL_PRESERVE_BLANKS=YES` or set `NULLCHARVAL` to a non-blank value and then replace the non-blank value with blanks after processing, if necessary.

z/OS Specifics

When you use bulk loading in the z/OS operating environment, the files that the SQL*Loader uses must conform to z/OS data set standards. The data sets can be either sequential data sets or partitioned data sets. Each filename that is supplied to the SQL*Loader are subject to extension and FNA processing.

If you do not specify filenames using data set options, default names in the form of `userid.SAS.data-set-extension` apply. The `userid` is the TSO prefix when running under TSO, and it is the PROFILE PREFIX in batch. The `data-set-extensions` are:

- BAD for the bad file
- CTL for the control file
- DAT for the data file
- DSC for the discard file
- LOG for the log file

If you want to specify filenames using data set options, you must use one of these forms:

- `/DD/ddname`
- `/DD/ddname(membername)`
- `Name`

For detailed information about these forms, see the SQL*Loader chapter in the Oracle user's guide for z/OS.

The Oracle engine runs the SQL*Loader by issuing a host-system command from within your SAS session. The data set where the SQLLDR executable file resides must be available to your TSO session or allocated to your batch job. Check with your system administrator if you do not know the name or availability of the data set that contains the SQLLDR executable file.

By default, on z/OS the bad file and discard file are not created in the same format as the data file. This makes it difficult to load the contents of these files after making corrections. See the section on SQL*Loader file attributes in the SQL*Loader section in the Oracle user's guide for z/OS for information about overcoming this limitation.
Example

This example shows you how to create and use a SAS data set to create and load to a large Oracle table, FLIGHTS98. This load uses the SQL*Loader direct path method because you specified BULKLOAD=YES. BL_OPTIONS= passes the specified SQL*Loader options to SQL*Loader when it is invoked. In this example, you can use the ERRORS= option to have up to 899 errors in the load before it terminates. Also, the LOAD= option loads the first 5,000 rows of the input data set, SASFLT.FLT98.

options yearcutoff=1926; /* included for Year 2000 compliance */
libname sasflt 'SAS-library';
libname ora_air oracle user=myusr1 password=mypwd1
   path='mysrv1_flt' schema=statsdiv;

data sasflt.flt98;
   input flight $3. +5 dates date7. +3 depart time5. +2 orig $3. +3 dest $3. +7 miles +6 boarded +6 capacity;
   format dates date9. depart time5.;
   informat dates date7. depart time5.;
   datalines;
   114     01JAN98    7:10  LGA   LAX       2475       172       210
   202     01JAN98   10:43  LGA   ORD        740       151       210
   219     01JAN98    9:31  LGA   LON       3442       198       250
<...10,000 more observations...>
proc sql;
   create table ora_air.flights98
      (BULKLOAD=YES BL_OPTIONS='ERRORS=899,LOAD=5000') as
      select * from sasflt.flt98;
   quit;

During a load, certain SQL*Loader files are created, such as the data, log, and control files. Unless otherwise specified, they are given a default name and written to the current directory. For this example, the default names would be bl_flights98.dat, bl_flights98.log, and bl_flights98.ctl.

Locking in the Oracle Interface

These LIBNAME and data set options let you control how the Oracle interface handles locking. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

READ_LOCK_TYPE= NOLOCK | ROW | TABLE

The default value is NOLOCK. Here are the valid values for this option:

• NOLOCK — table locking is not used during the reading of tables and views.

• ROW — the Oracle ROW SHARE table lock is used during the reading of tables and views.

• TABLE — the Oracle SHARE table lock is used during the reading of tables and views.
If you set READ_LOCK_TYPE= to either TABLE or ROW, you must also set the CONNECTION= option to UNIQUE. If not, an error occurs.

UPDATE_LOCK_TYPE= NOLOCK | ROW | TABLE
The default value is NOLOCK. Here are the valid values for this option:

- ROW — the Oracle ROW SHARE table lock is used during the reading of tables and views for update.
- TABLE — the Oracle EXCLUSIVE table lock is used during the reading of tables and views for update.
- NOLOCK — table locking is not used during the reading of tables and views for update.
  
  - If OR_UPD_NOWHERE=YES, updates are performed using serializable transactions.
  
  - If OR_UPD_NOWHERE=NO, updates are performed using an extra WHERE clause to ensure that the row has not been updated since it was first read. Updates might fail under these conditions, because other users might modify a row after the row was read for update.

READ_ISOLATION_LEVEL= READEMITTED | SERIALIZABLE
Oracle supports the READEMITTED and SERIALIZABLE read isolation levels, as defined in the following table. The SPOOL= option overrides the READ_ISOLATION_LEVEL= option. The READ_ISOLATION_LEVEL= option should be rarely needed because the SAS/ACCESS engine chooses the appropriate isolation level based on other locking options.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIALIZABLE</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads.</td>
</tr>
<tr>
<td>READEMITTED</td>
<td>Does not allow dirty Reads; does allow nonrepeatable Reads and phantom Reads</td>
</tr>
</tbody>
</table>

UPDATE_ISOLATION_LEVEL= READEMITTED | SERIALIZABLE
Oracle supports the READEMITTED and SERIALIZABLE isolation levels, as defined in the preceding table, for updates.

This option should be rarely needed because the SAS/ACCESS engine chooses the appropriate isolation level based on other locking options.

Naming Conventions for Oracle

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Oracle is case sensitive, and all names default to uppercase.
You can name such Oracle objects as tables, views, columns, and indexes. For the Oracle7 Server, objects also include database triggers, procedures, and stored functions. They follow these naming conventions.

- A name must be from 1 to 30 characters long. Database names are limited to 8 characters, and link names are limited to 128 characters.
- A name must begin with a letter. However, if you enclose the name in double quotation marks, it can begin with any character.
- A name can contain the letters A through Z, the digits 0 through 9, the underscore (_), $, and #. If the name appears within double quotation marks, it can contain any characters, except double quotation marks.
- Names are not case sensitive. For example, CUSTOMER and Customer are the same. However, if you enclose an object names in double quotation marks, it is case sensitive.
- A name cannot be an Oracle reserved word.
- A name cannot be the same as another Oracle object in the same schema.

Data Types for Oracle

Overview

Every column in a table has a name and a data type. The data type tells Oracle how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Oracle data types, null and default values, and data conversions.

Supported Oracle Data Types

Here are the data types that the Oracle engine supports.

- Character data:
  - CHAR (n)
  - NVARCHAR(n)
  - CLOB (character large object)
  - VARCHAR2(n)
  - NCHAR(n)

- Numeric data:
  - BINARY_DOUBLE
  - NUMBER(p)
  - BINARY_FLOAT
  - NUMBER(p,s)
  - NUMBER

Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.
• Date, timestamp, and interval data:

  - DATE, TIMESTAMP
  - INTERVAL YEAR TO MONTH
  - TIMESTAMP WITH TIME ZONE
  - INTERVAL DAY TO SECOND
  - TIMESTAMP WITH LOCAL TIME ZONE

• Binary data:

  - RAW(n) BLOB

SAS/ACCESS Interface to Oracle does not support the Oracle MLSLABEL data type. For compatibility with other DBMSs, Oracle supports the syntax for a wide variety of numeric data types, including DECIMAL, INTEGER, REAL, DOUBLE-PRECISION, and SMALLINT. All forms of numeric data types are actually stored in the same internal Oracle NUMBER format. The additional numeric data types are variations of precision and scale. A null scale implies a floating-point number, and a non-null scale implies a fixed-point number.

For detailed information about Oracle data types, see your Oracle documentation.

**Example 1: Timestamp**

```sas
%let PTCONN= %str(user=myusr1 pw=mypwd1 path=mysrv1);
%let CONN= %str(user=myusr1 pw=mypwd1 path=mysrv1);
options sastrace=",,," sastraceloc=saslog nostsuffix;
proc sql;
connect to oracle ( &PTCONN);
/* Execute ( drop table EMP_ATTENDANCE) by oracle;*/
execute ( create table EMP_ATTENDANCE (  EMP_NAME VARCHAR2(10),
arrival_timestamp TIMESTAMP, departure_timestamp TIMESTAMP ) ) by oracle;
execute ( insert into EMP_ATTENDANCE values
('John Doe', systimestamp, systimestamp+.2) ) by oracle;
execute ( insert into EMP_ATTENDANCE values
('Sue Day', TIMESTAMP'1980-1-12 10:13:23.33',
TIMESTAMP'1980-1-12 17:13:23.33' )) by oracle;
quit;
libname ora oracle &CONN
proc contents data=ora.EMP_ATTENDANCE; run;
proc sql;
/* Read TIMESTAMP data type */
select * from ora.EMP_ATTENDANCE;
quit;
/* Append to TIMESTAMP data type */
data work.new;
  EMP_NAME='New Bee1';
  ARRIVAL_TIMESTAMP='30sep1998:14:00:35.00'dt;
  DEPARTURE_TIMESTAMP='30sep1998:17:00:14.44'dt; output;
  EMP_NAME='New Bee2';
  ARRIVAL_TIMESTAMP='30sep1998:11:00:25.11'dt;
```
DEPARTURE_TIMESTAMP='30sep1998:14:00:35.27' dt; output;
EMP_NAME='New Bee3';
ARRIVAL_TIMESTAMP='30sep1998:08:00:35.33' dt;
DEPARTURE_TIMESTAMP='30sep1998:17:00:35.10' dt; output;
format ARRIVAL_TIMESTAMP datetime23.2;
format DEPARTURE_TIMESTAMP datetime23.2;
run;

title2 'After append';
proc append data=work.new base=ora.EMP_ATTENDANCE ; run;
proc print data=ora.EMP_ATTENDANCE ; run;
/* Update TIMESTAMP data type */
proc sql;
update ora.EMP_ATTENDANCE set ARRIVAL_TIMESTAMP=.
where EMP_NAME like '%Bee2%'
select * from ora.EMP_ATTENDANCE
delete from ora.EMP_ATTENDANCE where EMP_NAME like '%Bee2%'
select * from ora.EMP_ATTENDANCE

/* OUTPUT: Creating a brand new table using Data Step*/
data work.sasdsfsec; c_ts='30sep1998:14:00:35.16' dt; k=1; output;
c_ts='.' dt; k=2; output;
format c_ts datetime23.2; run;
/* picks default TIMESTAMP type */
options sastrace=",,d" sastraceloc=saslog nostsuffix;
data ora.tab_tsfsec; set work.sasdsfsec; run;
options sastrace=",," sastraceloc=saslog nostsuffix;
proc datasets library=ora;
delete tab_tsfsec;run;
/* Override the default data type */
options sastrace=",,d" sastraceloc=saslog nostsuffix;
data ora.tab_tsfsec (dbtype=(c_ts='timestamp(3)'));
c_ts='30sep1998:14:00:35' dt;
format c_ts datetime23.; run;
options sastrace=",," sastraceloc=saslog nostsuffix;
proc datasets library=ora;
delete tab_tsfsec;run;
proc print data=ora.tab_tsfsec; run;
/* Output: Create a new table with bulkload=yes */
title2 'Test OUTPUT with bulkloader';
proc datasets library=ora;
delete tab_tsfsec;run;
/* Select default TIMESTAMP type */
data ora.tab_tsfsec (bulkload=yes); set work.sasdsfsec; run;
proc print data=ora.tab_tsfsec; run;
Example 2: Interval Year to Month

```sql
proc sql;
connect to oracle ( &PTCONN);
execute ( drop table PRODUCT_INFO) by oracle;
execute (
    create table PRODUCT_INFO (  PRODUCT VARCHAR2(20),
        LIST_PRICE number(8,2),
        WARRANTY_PERIOD INTERVAL YEAR(2) TO MONTH )) by oracle;
execute (
    insert into PRODUCT_INFO values ('Dish Washer', 4000, '02-00')
) by Oracle;
execute (
    insert into PRODUCT_INFO values ('TV', 6000, '03-06')) by Oracle;
quit;
```

```sql
proc contents data=ora.PRODUCT_INFO;run;

/* Show WARRANTY_PERIOD as number of months */
proc print data=ora.PRODUCT_INFO; run;

/* Show WARRANTY_PERIOD in a format as in Oracle*/
proc print
data=ora.PRODUCT_INFO(dbsastype=(WARRANTY_PERIOD='CHAR(6)')); run;

/* Add a new product */
data new_prods;
    PRODUCT='Dryer';  LIST_PRICE=2000;WARRANTY_PERIOD=12;
run;

proc sql;
insert into ora.PRODUCT_INFO select * from new_prods;
select * from ora.PRODUCT_INFO;
select * from ora.PRODUCT_INFO where WARRANTY_PERIOD > 24;
quit;
```

Example 3: Interval Day to Second

```sql
proc sql;
connect to oracle ( &PTCONN);
execute ( drop table PERF_TESTS) by oracle;
execute (
    create table PERF_TESTS ( TEST_NUMBER number(4) primary key,
        TIME_TAKEN INTERVAL DAY TO SECOND )) by oracle;
execute (
    insert into PERF_TESTS
        values (1, '0 00:01:05.000200000') by Oracle;
execute (
    insert into PERF_TESTS values (2, '0 00:01:03.400000000')) by Oracle;
quit;
```
Default Data Types

Oracle Null and Default Values
Oracle has a special value called NULL. An Oracle NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads an Oracle NULL value, it interprets it as a SAS missing value.

By default, Oracle columns accept NULL values. However, you can define columns so that they cannot contain NULL data. NOT NULL tells Oracle not to add a row to the table unless the row has a value for that column. When creating an Oracle table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

To control how SAS missing character values are handled, use the NULLCHAR= and NULLCHARVAL= data set options.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

Default Data Types for SAS Output
The following table shows the default data types for SAS character variables based on the length of the variable and the Oracle version:

<table>
<thead>
<tr>
<th>Oracle Server</th>
<th>Character Variable Length</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior to 12c</td>
<td>less than or equal to 4000</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>greater than 4000</td>
<td>CLOB</td>
</tr>
<tr>
<td>12c and later</td>
<td>less than or equal to 32767</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>greater than 32767</td>
<td>CLOB</td>
</tr>
</tbody>
</table>
The following table shows the default data types for SAS character variables when the NOTRANSCODE attribute is set:

<table>
<thead>
<tr>
<th>Oracle Server</th>
<th>Character Variable Length</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior to 12c</td>
<td>less than or equal to 2000</td>
<td>RAW</td>
</tr>
<tr>
<td></td>
<td>greater than 2000</td>
<td>BLOB</td>
</tr>
<tr>
<td>12c and later</td>
<td>less than or equal to 32767</td>
<td>RAW</td>
</tr>
<tr>
<td></td>
<td>greater than 32767</td>
<td>BLOB</td>
</tr>
</tbody>
</table>

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to Oracle assigns to SAS variables when using the **LIBNAME statement** to read from an Oracle table. These default formats are based on Oracle column attributes.

**Table 28.4 LIBNAME Statement: Default SAS Formats for Oracle Data Types**

<table>
<thead>
<tr>
<th>Oracle Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n) *</td>
<td>$w.</td>
</tr>
<tr>
<td>NCHAR(n) *</td>
<td>$w.</td>
</tr>
<tr>
<td>NVARCHAR(n) *</td>
<td>$w.</td>
</tr>
<tr>
<td>VARCHAR2(n)</td>
<td>$w.</td>
</tr>
<tr>
<td>LONG</td>
<td>$w. (where w is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>CLOB</td>
<td>$w. (where w is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>RAW(n)</td>
<td>$HEXw.$ (where w is 2*n)</td>
</tr>
<tr>
<td>LONG RAW</td>
<td>$HEXw. (where w/2 is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>BLOB RAW</td>
<td>$HEXw. (where w/2 is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>BINARY_DOUBLE</td>
<td>none</td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>none</td>
</tr>
<tr>
<td>NUMBER</td>
<td>none</td>
</tr>
</tbody>
</table>
**Oracle Data Type** | **Default SAS Format**
---|---
NUMBER \((p)\) | \(w.\)
NUMBER \((p,s)\) | \(w.d\)
DATE | DATETIME20.
TIMESTAMP | DATETIME\(w.d\) (where \(d\) is derived from the fractional-second precision)
TIMESTAMP WITH LOCAL TIMEZONE | DATETIME\(w.d\) (where \(d\) is derived from the fractional-second precision)
TIMESTAMP WITH TIMEZONE | \(sw.\)
INTERVAL YEAR TO MONTH | \(w.\) (where \(w\) is derived from the year precision)
INTERVAL DAY TO SECOND | \(w.d\) (where \(w\) is derived from the fractional-second precision)

* The value of the “DBMAX_TEXT= LIBNAME Option” option can override these values.

SAS/ACCESS does not support Oracle data types that do not appear in this table.

If Oracle data falls outside valid SAS data ranges, the values are usually counted as missing.

SAS automatically converts Oracle NUMBER types to SAS number formats by using an algorithm that determines the correct scale and precision. When the scale and precision cannot be determined, SAS/ACCESS allows the procedure or application to determine the format. You can also convert numeric data to character data by using the SQL pass-through facility with the Oracle TO_CHAR function. See your Oracle documentation for more details.

The following table shows the default Oracle data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

**Table 28.5 LIBNAME Statement: Default Oracle Data Types for SAS Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Oracle Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(sw.)</td>
<td>VARCHAR2 ((w))</td>
</tr>
<tr>
<td>(sw.) (where (w \geq 4000))</td>
<td>CLOB</td>
</tr>
<tr>
<td>(w.d)</td>
<td>NUMBER ((p,s))</td>
</tr>
<tr>
<td>any date, time, or datetime format without fractional parts of a second</td>
<td>DATE</td>
</tr>
<tr>
<td>any date, time, or datetime format without fractional parts of a second</td>
<td>TIMESTAMP</td>
</tr>
</tbody>
</table>
To override these data types, use the DBTYPE= data set option during output processing.

**ACCESS Procedure Data Conversions**

This table shows the default SAS variable formats that SAS/ACCESS assigns to Oracle data types when you use the ACCESS procedure.

*Table 28.6  PROC ACCESS: Default SAS Formats for Oracle Data Types*

<table>
<thead>
<tr>
<th>Oracle Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR((n))</td>
<td>(n. (n &lt;= 200) \quad 200. (n &gt; 200))</td>
</tr>
<tr>
<td>VARCHAR2((n))</td>
<td>(n. (n &lt;= 200) \quad 200. (n &gt; 200))</td>
</tr>
<tr>
<td>FLOAT</td>
<td>BEST22.</td>
</tr>
<tr>
<td>NUMBER</td>
<td>BEST22.</td>
</tr>
<tr>
<td>NUMBER((p))</td>
<td>(w.)</td>
</tr>
<tr>
<td>NUMBER((p, s))</td>
<td>(w.d)</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME16.</td>
</tr>
<tr>
<td>CLOB</td>
<td>(200.)</td>
</tr>
<tr>
<td>RAW((n))</td>
<td>(n. (n &lt; 200) \quad 200. (n &gt; 200))</td>
</tr>
<tr>
<td>BLOB RAW</td>
<td>(200.)</td>
</tr>
</tbody>
</table>

Oracle data types that are omitted from this table are not supported by SAS/ACCESS. If Oracle data falls outside valid SAS data ranges, the values are usually counted as missing.

This table shows the correlation between the Oracle NUMBER data types and the default SAS formats that are created from that data type.

*Table 28.7  Default SAS Formats for Oracle NUMBER Data Types*

<table>
<thead>
<tr>
<th>Oracle NUMBER Data Type</th>
<th>Rules</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER((p))</td>
<td>(0 &lt; p &lt;= 32)</td>
<td>((p + 1).0)</td>
</tr>
<tr>
<td>NUMBER((p, s))</td>
<td>(p &gt; 0, s &lt; 0,</td>
<td>s</td>
</tr>
<tr>
<td>NUMBER((p, s))</td>
<td>(p &gt; 0, s &lt; 0,</td>
<td>s</td>
</tr>
<tr>
<td>NUMBER((p, s))</td>
<td>(p &gt; 0, s &gt; 0, s &lt; p)</td>
<td>((p + 2).s)</td>
</tr>
<tr>
<td>NUMBER((p, s))</td>
<td>(p &gt; 0, s &gt; 0, s &gt;= p)</td>
<td>((s + 3).s)</td>
</tr>
</tbody>
</table>
Oracle NUMBER Data Type  |  Rules  |  Default SAS Format  
---|---|---
NUMBER(p)  |  \( p > 32 \)  |  BEST22. SAS selects format  
NUMBER  |  \( p, s \) unspecified  |  BEST22. SAS selects format  

The general form of an Oracle number is \( \text{NUMBER}(p,s) \) where \( p \) is the precision and \( s \) is the scale of the number. Oracle defines precision as the total number of digits, with a valid range of \(-84\) to \(127\). However, a negative scale means that the number is rounded to the specified number of places to the left of the decimal. For example, if the number \(1,234.56\) is specified as data type \( \text{NUMBER}(8, -2) \), it is rounded to the nearest hundred and stored as \(1,200\).

**DBLOAD Procedure Data Conversions**

This table shows the default Oracle data types that SAS/ACCESS assigns to SAS variable formats when you use the DBLOAD procedure.

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Oracle Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.</td>
<td>CHAR((n))</td>
</tr>
<tr>
<td>w.</td>
<td>( \text{NUMBER}(p) )</td>
</tr>
<tr>
<td>w.d</td>
<td>( \text{NUMBER}(p,s) )</td>
</tr>
<tr>
<td>all other numerics *</td>
<td>( \text{NUMBER} )</td>
</tr>
<tr>
<td>datetime w.d</td>
<td>DATE</td>
</tr>
<tr>
<td>date w.</td>
<td>DATE</td>
</tr>
<tr>
<td>time. **</td>
<td>( \text{NUMBER} )</td>
</tr>
</tbody>
</table>

* Includes all SAS numeric formats, such as BINARY8 and E10.0.
** Includes all SAS time formats, such as TODw,d and HHMMw,d.
Chapter 29
SAS/ACCESS Interface to PostgreSQL

Introduction to SAS/ACCESS Interface to PostgreSQL ........................................... 811
LIBNAME Statement Specifics for PostgreSQL ...................................................... 812
  Overview .................................................................................. 812
  Arguments ................................................................................. 812
  PostgreSQL LIBNAME Statement Examples ......................................................... 815
Data Set Options for PostgreSQL ......................................................................... 815
SQL Pass-Through Facility Specifics for PostgreSQL ........................................ 817
  Key Information ........................................................................... 817
  CONNECT Statement Examples ............................................................... 818
  Working with Long Character Values in PostgreSQL ........................................ 818
Passing SAS Functions to PostgreSQL ................................................................. 819
Passing Joins to PostgreSQL ............................................................................... 820
Bulk Loading for PostgreSQL ............................................................................. 820
  Overview .................................................................................. 820
  Data Set Options with Bulk Loading ................................................................. 820
  Using the PSQL Tool for Bulk Loading ............................................................. 821
  Examples .................................................................................... 821
Locking in the PostgreSQL Interface ................................................................... 821
Naming Conventions for PostgreSQL ................................................................. 822
Data Types for PostgreSQL ................................................................................. 823
  Overview .................................................................................. 823
  PostgreSQL Null Values ........................................................................... 823
  LIBNAME Statement Data Conversions ......................................................... 823

Introduction to SAS/ACCESS Interface to PostgreSQL

For available SAS/ACCESS features, see “SAS/ACCESS Interface to PostgreSQL: Supported Features” on page 87. For more information about PostgreSQL, see your PostgreSQL documentation.

Note: The SAS/ACCESS Interface to PostgreSQL was implemented for SAS 9.4.
LIBNAME Statement Specifics for PostgreSQL

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to PostgreSQL supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing PostgreSQL.

LIBNAME libref postgres <connection-options> <LIBNAME-options>;

Arguments

libref
specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

postgres
specifies the SAS/ACCESS engine name for the PostgreSQL interface.

connection-options
provide connection information and control how SAS manages the timing and concurrency of the connection to the DBMS. Here is how these options are defined.

SERVER=<’PostgreSQL-server-name’>
specifies the server name or IP address of the PostgreSQL server to which you want to connect. This server accesses the database that contains the tables and views that you want to access. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

DATABASE=<’PostgreSQL-database-name’>
specifies the name of the database on the PostgreSQL server that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: DB=

PORT=port
specifies the port number that is used to connect to the specified PostgreSQL server.

Default: 5432

USER=<’PostgreSQL-user-name’>
specifies the PostgreSQL user name (also called the user ID) that you use to connect to your database. If the user name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

PASSWORD=<’PostgreSQL-password’>
specifies the password that is associated with your PostgreSQL user name. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: PASS=, PW=, PWD=
DSN=<"PostgreSQL-data-source"> specifies the configured PostgreSQL ODBC data source to which you want to connect. Use this option if you have existing PostgreSQL ODBC data sources that are configured on your client. This method requires additional setup—either through the Administrator control panel on Windows platforms or through the odbc.ini file or a similarly named configuration file on UNIX platforms. So it is recommended that you use this connection method only if you have existing, functioning data sources that have been defined.

**LIBNAME-options** define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to PostgreSQL with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

**Table 29.1 SAS/ACCESS LIBNAME Options for PostgreSQL**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>NO</td>
</tr>
<tr>
<td>BL_DEFAULT_DIR=</td>
<td>none</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCLIENT_MAX_BYTES=</td>
<td>matches the maximum number of bytes per single character of the SAS session encoding</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>1</td>
</tr>
<tr>
<td>LOGIN_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>YES</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>YES</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the PostgreSQL Interface” on page 821)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>0</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the PostgreSQL Interface” on page 821</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>USE_ODBC_CL=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>

**PostgreSQL LIBNAME Statement Examples**

In this example, SERVER=, DATABASE=, USER=, and PASSWORD= are connection options. No DSN style is specified. This is the default method, which is recommended.

```
libname A1 postgres server=mysrv1 port=5432
   user=myusr1 password='mypwd1' database=mydb1;
```

This example requires that you specify a DSN style.

```
libname B1 postgres dsn=ptgtest user=myusr1 password=mypwd1;
```

**Data Set Options for PostgreSQL**

All SAS/ACCESS data set options in this table are supported for PostgreSQL. Default values are provided where applicable. For more information, see Data Set Options for Relational Databases on page 241.

**Table 29.2  SAS/ACCESS Data Set Options for PostgreSQL**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DEFAULT_DIR=</td>
<td>&lt;'current-directory'&gt;</td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE=</td>
<td>YES</td>
</tr>
<tr>
<td>BL_DELIMITER=</td>
<td>, (comma)</td>
</tr>
<tr>
<td>BL_ESCAPE=</td>
<td>\</td>
</tr>
<tr>
<td>BL_FORMAT=</td>
<td>TEXT</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>BL_LOAD_METHOD=</code></td>
<td>APPEND (table with data)</td>
</tr>
<tr>
<td><code>BL_LOGFILE=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>BL_NULL=</code></td>
<td>'&quot;N' [TEXT mode], unquoted empty value [CSV mode]</td>
</tr>
<tr>
<td><code>BL_PSQL_PATH=</code></td>
<td>psql</td>
</tr>
<tr>
<td><code>BL_QUOTE=</code></td>
<td>&quot; (double quotation mark)</td>
</tr>
<tr>
<td><code>BULKLOAD=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBCOMMIT=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBCONDITION=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBCREATE_TABLE_OPTS=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBFORCE=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBGEN_NAME=</code></td>
<td>DBMS</td>
</tr>
<tr>
<td><code>DBINDEX=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBKEY=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBLABEL=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBMASTER=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBMAX_TEXT=</code></td>
<td>1024</td>
</tr>
<tr>
<td><code>DBNULL=</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>DBNULLKEYS=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBPROMPT=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBSASLABEL=</code></td>
<td>COMPAT</td>
</tr>
<tr>
<td><code>DBASTYPE=</code></td>
<td>see “Data Types for PostgreSQL” on page 823</td>
</tr>
<tr>
<td><code>DBTYPE=</code></td>
<td>see “Data Types for PostgreSQL” on page 823</td>
</tr>
<tr>
<td><code>ERRLIMIT=</code></td>
<td>1</td>
</tr>
<tr>
<td><code>IGNORE_READ_ONLY_COLUMNS=</code></td>
<td>NO</td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for PostgreSQL

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the PostgreSQL interface.

- The dbms-name is **POSTGRES**.
- The CONNECT statement is required.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT_SQL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>
PROC SQL supports multiple connections to PostgreSQL. If you use multiple simultaneous connections, you must use the alias argument to identify the different connections. If you do not specify an alias, the default POSTGRES alias is used.

The CONNECT statement database-connection-arguments are identical to its LIBNAME connection options.

**CONNECT Statement Examples**

This example connects to PostgreSQL and then disconnects from it.

```sql
proc sql noerrorstop;
   connect to postgres as x1(server=mysrv1 port=5432
      user=mysur1 password='mypwd1' database=mydb1);
   disconnect from x1;
quit;
```

This next example connects to PostgreSQL, executes some SQL statements, and then disconnects from PostgreSQL.

```sql
proc sql noerrorstop;
   connect to postgres as x1(server=mysrv1 port=5432
      user=mysur1 password='mypwd1' database=mydb1);
   execute ( CREATE TABLE t1 ( no int primary key, state varchar(10) ) ) by x1;
   execute ( INSERT INTO t1 values (1, 'USA') ) by x1;
   execute ( INSERT INTO t1 values (2, 'CHN') ) by x1;
   select * from connection to x1 (SELECT * FROM t1 ORDER BY no);
   disconnect from x1;
quit;
```

**Working with Long Character Values in PostgreSQL**

The PostgreSQL open-source driver automatically assigns a length of 255 characters to varying length character variables, such as the VARCHAR(n) data type. Therefore, if you know that character data in a data set is longer than 255 characters, you should use the MaxVarcharSize attribute when you create the connection to your PostgreSQL library. For example, the following LIBNAME statement and SQL query would create a new data set that does not truncate the character data from the original data set.

```sql
libname mfgapp postgres preserve_col_names=no
   preserve_tab_names=no database=mydb1 dbmax_text=32767
   server='myserver' port=5432 user=pgadmin password='pgpwd'
   conopts='MaxVarcharSize=32000;';
proc sql;
   create table child_analysis_temp2 as select
      case when IS_LONG_VALUE_LIST_FLG eq "1" then x.DSFILTER_VALUE_TXT
      else a.DSFILTER_VALUE_TXT end as dsfilter_value length=32000
      format=$32000.
      informat=$32000.
      from wrnaout.child_filter_1 a , wrnaout.child_filter_value_1 x
      where (a.Child_id = x.Child_id and
      a.dsfilter_column_nm=x.dsfilter_column_nm) and
      a.Child_id ="APHTS42CWF" ;
quit;
```
Passing SAS Functions to PostgreSQL

SAS/ACCESS Interface to PostgreSQL passes the following SAS functions to PostgreSQL for processing. Where the PostgreSQL function name differs from the SAS function name, the PostgreSQL name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

ABS
ARCOS (ACOS)
ARCSIN (ASIN)
ATAN
AVG
BYTE
CEIL
COALESCE
COMPRESS (TRANSLATE)
COS
COT
COUNT
DAY
DAYOFWEEK
EXP
FLOOR
HOUR
INDEX (POSITION, STRPOS)
LENGTH
LENGTHC (CHARACTER_LENGTH)
LENGTHN (LENGTH)
LOG (LN)
LOG10 (LOG)

LOG2 (LOG)
LOWCASE (LOWER)
MAX
MIN
MINUTE
MOD (see note)
MONTH
QTR
REPEAT
SECOND
SIGN
SIN
SQRT
STD (STDDEV_SAMP)
STRIP (BTRIM)
SUBSTR
SUM
TAN
TRANWRD (REPLACE)
TRIMN (RTRIM)
UPCASE (UPPER)
VAR (VAR_SAMP)
YEAR

Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to PostgreSQL. Due to incompatibility in date and time functions between PostgreSQL and SAS, PostgreSQL might not process them correctly. Check your results to determine whether these functions are working as expected.

ATAN2
DATE (current_date)
DATEPART (cast)
DATETIME (now)

ROUND
TIME (current_time)
TIMEPART (cast)
TODAY (current_date)
Passing Joins to PostgreSQL

For a multiple libref join to pass to PostgreSQL, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- server (SERVER=)
- database (DATABASE=)
- port (PORT=)
- SQL functions (SQL_FUNCTIONS=)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for PostgreSQL

Overview

Bulk loading is the fastest way to insert large numbers of rows into a PostgreSQL table. To use the bulk-load facility, specify the BULKLOAD=YES LIBNAME option. The bulk-load facility uses the PostgreSQL PSQL tool to move data from the client to the PostgreSQL database.

Data Set Options with Bulk Loading

Here are the PostgreSQL bulk-load data set options. For detailed information about these options, see “Overview” on page 241.

- BL_DATAFILE=
- BL_DEFAULT_DIR=
- BL_DELETE_DATAFILE=
- BL_DELIMITER=
- BL_ESCAPE=
- BL_FORMAT=
- BL_LOAD_METHOD=
- BL_LOGFILE=
- BL_NULL=
- BL_PSQL_PATH=
- BL_QUOTE=
Using the PSQL Tool for Bulk Loading

Required for bulk loading, the PSQL tool is a terminal-based front end to PostgreSQL. You can use it to enter queries interactively, submit them to PostgreSQL, and see the query results. You can also submit a file as input. PSQL provides a number of metacommands and various shell-like features to facilitate writing scripts and automating various tasks. It is available at this website: http://www.postgresql.org.

Examples

This first example shows how you can use a SAS data set, SASFLT.FLT98, to create and load a large PostgreSQL table, FLIGHTS98:

```sql
libname sasflt 'SAS-library';
libname net_air postgres user=myusr1 pwd=mypwd1
   server=air2 database=flights;

proc sql;
create table net_air.flights98 (bulkload=YES bl_psql_path='full path of PSQL')
   as select * from net_air.flt98;
quit;
```

Locking in the PostgreSQL Interface

The following LIBNAME and data set options let you control how the PostgreSQL interface handles locking. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

- **READ_LOCK_TYPE= ROW**
- **UPDATE_LOCK_TYPE= ROW**
- **READ_ISOLATION_LEVEL= S | RC**

The PostgreSQL ODBC driver manager supports the S and RC isolation levels that are defined in this table.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (serializable)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads.</td>
</tr>
<tr>
<td>RR (repeatable read)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RC (read committed)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>Isolation Level</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>RU (read uncommitted)</td>
<td>Allows dirty Reads, nonrepeatable Reads, and phantom Reads,</td>
</tr>
<tr>
<td>V (versioning)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads. These transactions are serializable but higher concurrency is possible than with the serializable isolation level. Typically, a nonlocking protocol is used.</td>
</tr>
</tbody>
</table>

Here are how the terms in the table are defined.

**Dirty reads**
A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it can see changes that are made by those concurrent transactions even before they commit.

For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.

**Phantom reads**
When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist, a phantom.

**UPDATE_ISOLATION_LEVEL= S | RC**
The PostgreSQL ODBC driver manager supports the S and RC isolation levels that are defined in the preceding table.

---

**Naming Conventions for PostgreSQL**

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The PostgreSQL interface supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name would result in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) PostgreSQL is not case sensitive, and all names default to lowercase.

PostgreSQL objects include tables, views, and columns. They follow these naming conventions.
Data Types for PostgreSQL

Overview

Every column in a table has a name and a data type. The data type tells PostgreSQL how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about PostgreSQL data types, null and default values, and data conversions.

For more information about PostgreSQL data types and to determine which data types are available for your version of PostgreSQL, see your PostgreSQL documentation.

PostgreSQL Null Values

PostgreSQL has a special value called NULL. A PostgreSQL NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a PostgreSQL NULL value, it interprets it as a SAS missing value.

You can define a column in a PostgreSQL table so that it requires data. To do this in SQL, you specify a column as NOT NULL. This tells SQL to allow a row to be added to a table only if a value exists for the field. For example, NOT NULL assigned to the CUSTOMER field in the SASDEMO.CUSTOMER table does not allow a row to be added unless there is a value for CUSTOMER. When creating a table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

You can also define PostgreSQL columns as NOT NULL DEFAULT. For more information about using the NOT NULL DEFAULT value, see your PostgreSQL documentation.

Once you know whether a PostgreSQL column enables NULLs or the host system supplies a default value for a column that is defined as NOT NULL WITH DEFAULT, you can write selection criteria and enter values to update a table. Unless a column is defined as NOT NULL or NOT NULL DEFAULT, it allows NULL values.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how SAS missing character values are handled, use the NULLCHAR= and NULLCHARVAL= data set options.

LIBNAME Statement Data Conversions

This table shows the default formats that SAS/ACCESS Interface to PostgreSQL assigns to SAS variables when using the LIBNAME statement to read from a PostgreSQL table.
These default formats are based on PostgreSQL column attributes. SAS/ACCESS does not support PostgreSQL data types that do not appear in this table.

**Table 29.4 LIBNAME Statement: Default SAS Formats for PostgreSQL Data Types**

<table>
<thead>
<tr>
<th>PostgreSQL Data Type</th>
<th>ODBC Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR((n)) *</td>
<td>SQL_CHAR</td>
<td>$w.</td>
</tr>
<tr>
<td>VARCHAR((n)) *</td>
<td>SQL_LONGCHAR((n))</td>
<td></td>
</tr>
<tr>
<td>BYTEA</td>
<td>SQL_BINARY</td>
<td>$w***</td>
</tr>
<tr>
<td></td>
<td>SQL_VARBINARY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SQL_LONGVARBINARY</td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td>SQL_DECIMAL</td>
<td>(w) or (w.d) or none if you do not specify (w) and (d)**</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>SQL_NUMERIC</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>SQL_INTEGER</td>
<td>11.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SQL_SMALLINT</td>
<td>6.</td>
</tr>
<tr>
<td></td>
<td>SQL_TINYINT</td>
<td>4.</td>
</tr>
<tr>
<td>BIT(1)</td>
<td>SQL_BIT</td>
<td>1.</td>
</tr>
<tr>
<td>REAL</td>
<td>SQL_REAL</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT</td>
<td>SQL_FLOAT</td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>SQL_DOUBLE</td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>SQL_BIGINT</td>
<td>20.</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>SQL_INTERVAL</td>
<td>$w.</td>
</tr>
<tr>
<td>UUID</td>
<td>SQL_GUID</td>
<td>$w.</td>
</tr>
<tr>
<td>DATE</td>
<td>SQL_TYPE_DATE</td>
<td>DATE9.</td>
</tr>
</tbody>
</table>

\* \(n\) in PostgreSQL character data types is equivalent to \(w\) in SAS formats.

\** \(m\) and \(n\) in PostgreSQL numeric data types are equivalent to \(w\) and \(d\) in SAS formats.

\*** Because the Postgres ODBC driver does the actual conversion, this field is displayed as if the $HEX\(w\). format were applied.

The following table shows the default data types that SAS/ACCESS Interface to PostgreSQL uses when creating tables. The PostgreSQL engine lets you specify nondefault data types by using the DBTYPE= data set option.
Table 29.5  LIBNAME Statement: Default SAS Formats for PostgreSQL Data Types When Creating Tables

<table>
<thead>
<tr>
<th>PostgreSQL Data Type</th>
<th>ODBC Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC(m,n)**</td>
<td>SQL_DOUBLE or</td>
<td>w.d</td>
</tr>
<tr>
<td></td>
<td>SQL_NUMERIC using m,n if</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the DBMS allows it</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n)*</td>
<td>SQL_VARCHAR using n</td>
<td>$w.$</td>
</tr>
<tr>
<td>TIMESTAMP***</td>
<td>SQL_TIMESTAMP</td>
<td>DATETIME formats</td>
</tr>
<tr>
<td>DATE</td>
<td>SQL_DATE</td>
<td>DATE formats</td>
</tr>
<tr>
<td>TIME</td>
<td>SQL_TIME</td>
<td>TIME formats</td>
</tr>
</tbody>
</table>

*  n in PostgreSQL character data types is equivalent to w in SAS formats.
** m and n in PostgreSQL numeric data types are equivalent to w and d in SAS formats.
*** Although the PostgreSQL engine supports TIMESTAMP, it has no TIMESTAMP WITH TIMEZONE data type that maps to the corresponding Postgres data type.
Chapter 30
SAS/ACCESS Interface to SAP HANA

Introduction to SAS/ACCESS Interface to SAP HANA ............................... 828
LIBNAME Statement Specifics for SAP HANA ....................................... 828
Overview ......................................................................................... 828
Arguments ...................................................................................... 828
SAP HANA LIBNAME Statement Examples ....................................... 833
Data Set Options for SAP HANA ....................................................... 834
SQL Pass-Through Facility Specifics for SAP HANA ............................... 836
Key Information ............................................................................... 836
CONNECT Statement Examples ...................................................... 836
Understanding SAP HANA Update and Delete Rules .............................. 837
Autopartitioning Scheme for SAP HANA ........................................... 838
Overview ......................................................................................... 838
Autopartitioning Restrictions ............................................................ 838
Nullable Columns .......................................................................... 838
Using WHERE Clauses ..................................................................... 838
Using DBSLICEPARAM= ................................................................. 838
Using DBSLICE= ............................................................................ 838
SAP HANA Schema Flexibility ......................................................... 839
Passing SAS Functions to SAP HANA ............................................... 839
Passing Joins to SAP HANA .............................................................. 840
Bulk Loading for SAP HANA ............................................................ 841
Overview ......................................................................................... 841
Data Set Options forBulk Loading .................................................... 841
Naming Conventions for SAP HANA ............................................... 841
Data Types for SAP HANA .............................................................. 842
Overview ......................................................................................... 842
Data Types for SAP HANA .............................................................. 842
Working with Geospatial Data in SAS ............................................... 844
LIBNAME Statement Data Conversions ............................................ 846
Introduction to SAS/ACCESS Interface to SAP HANA

For available SAS/ACCESS features, see SAP HANA supported features on page 88. For more information about SAP HANA, see your SAP HANA documentation.

Note: The SAS/ACCESS Interface to SAP HANA was implemented for SAS 9.4.

Note: When you use analytic views in SQL statements, the measures must be aggregated. Beginning in the third maintenance release of SAS 9.4, the SAS/ACCESS engine generates a default statement with aggregated measures based on the metadata about the analytic view.

In the third maintenance release of SAS 9.4, PARMSTRING= and PARMDEFAULT= LIBNAME options and PARMSTRING= and PARMDEFAULT= data set options are available. The PARMSTRING= options specify a quoted string of variable name and value pairs or a placeholder string. The PARMDEFAULT= options specify whether the SAP HANA engine should use the defaults for variables and parameters as specified in the metadata in SAP HANA. For more information, see the following documentation:

- “PARMSTRING= LIBNAME Option” on page 181
- “PARMSTRING= Data Set Option” on page 389
- “PARMDEFAULT= LIBNAME Option” on page 181
- “PARMDEFAULT= Data Set Option” on page 389

LIBNAME Statement Specifics for SAP HANA

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to SAP HANA supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing SAP HANA.

LIBNAME libref saphana <connection-options> <LIBNAME-options>;

Arguments

libref
specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

saphana
specifies the SAS/ACCESS engine name for the SAP HANA interface.

Note: Support for the saphana engine name was added in the second maintenance release for SAS 9.4.
connection-options

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to the SAP HANA server in many different ways. Specify only one of these methods for each connection because they are mutually exclusive.

- SERVER=, PORT=, USER=, PASSWORD=
- SERVER=, INSTANCE=, USER=, PASSWORD=
- DSN=, USER=, PASSWORD=[This method uses a configured ODBC data source to store the connections options.]
- NOPROMPT=
- PROMPT=

Here is how these options are defined.

SERVER=<server-name>
SERVER=<<server-name:port>>
SERVER='server-name:port;failover-server-name1:port;failover-server-name2:port'

specifies the server name or IP address of the SAP HANA server to which you want to connect. If the server name contains spaces or nonalphanumeric characters or if it is an IP address, you must enclose it in quotation marks. You can include the port when you specify a server. The port number is $3<instance-number>15$ (for example, 30015 for instance 00). To support failover, you can specify a list of hostnames, separated by a semicolon. If a host is not available, the next host from the list is used.

Alias: HOST, SERVER, SERVERNODE

Example: SERVERNODE='saph1.mycompany.com'

PORT=port

specifies the port number that is used to connect to the specified SAP HANA server. The default is used if you do not specify a port or instance number or if you do not include the port number in the server specification. The port for the standard SQL communication for client access is $3<instance-number>15$.

Restriction: Do not specify a value for both the PORT= and INSTANCE= options. If you specify a value for both PORT= and INSTANCE=, then the following error appears in the log:

ERROR: Invalid combination of connection errors. You cannot use the PORT with the INSTANCE option.

Alias: PORT

Default: 30015

Example: server='saph1.mycompany.com' port=30215

INSTANCE=instance-number

specifies the instance number of the SAP HANA database engine. The port number is $3<instance-number>15$ (for example, 30015 for instance 00). If you specify the port number explicitly in either PORT= or SERVER=, then INSTANCE= is ignored and a warning is written to the SAS log.

Restriction: Do not specify INSTANCE= if you also specify a port value for PORT= or SERVER=. If you specify a value for both PORT= and INSTANCE=, then the following error appears in the log:

ERROR: Invalid combination of connection errors. You cannot use the
PORT with the INSTANCE option.

Alias: INSTANCE

Example: `server='saph1.mycompany.com' instance=02`

USER=<'SAP-HANA-user-name'>

specifies the SAP HANA user name (also called the user ID) that you use to connect to your database. If the user name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: USER

Example: `USER=HANAUSER1`

PASSWORD=<'SAP-HANA-password'>

specifies the password that is associated with your SAP HANA user name. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Aliases: PASS, PASSWORD, PW, PWD

DSN=<'SAP-HANA-data-source'>

specifies the configured SAP HANA ODBC data source to which you want to connect. Use this option if you have existing SAP HANA ODBC data sources that are configured on your client. This method requires additional setup. You can perform setup either through the ODBC Administrator control panel on Windows platforms or through the odbc.ini file or a similarly named configuration file on UNIX platforms. It is recommended that you use this connection method only if you have existing, functioning data sources that have been defined.

Alias: DSN

Example: `DSN=HANADSN1`

Here is an example for an odbc.ini entry on UNIX:

```
[MYSAPHANA]
SERVERNODE=saph1.mycompany.com:30315
```

NOPROMPT=<'SAP-HANA-connection-options'>

specifies connection options for your data source or database. Separate multiple options with a semicolon. If you do not specify enough correct connection options, an error is returned. No dialog box is displayed to help you complete the connection string. Connection options are appended to the connection string that is used to connect to the SAP HANA database. You can use this option to specify special options by adding key-value pairs to the connection string.

PROMPT=<'SAP-HANA-connection-information'>

specifies connection options for your data source or database. Separate multiple options with a semicolon. When connection succeeds, the complete connection string is returned in the SYSDBMGS macro variable. PROMPT= does not immediately try to connect to the DBMS. A dialog box is displayed instead that contains the values that you entered in the PROMPT= connection string. You can edit values or enter additional values in any field before you connect to the data source. This option is not supported on UNIX platforms.

Restriction: not supported on UNIX platforms

DRIVER=<'driver'>

specifies the ODBC driver to use to connect to your data source or database.
ENCRIPT=YES | NO  
   specifies how communication is encrypted.  
   Alias: ENCRYPT  
   Default: NO

SSLCRYPTOPROVIDER=SAPCRYPTO | OPENSSL | MSCRYPTO  
   specifies the cryptographic library provider to use for SSL connectivity.  
   Alias: SSLCRYPTOPROVIDER, SSLPROVIDER

SSLKEYSTORE=<'file'>  
   specifies the path to the keystore file.  
   Alias: SSLKEYSTORE  
   ODBC driver default: $HOME/.ssl/key.pem

SSLTRUSTSTORE=<'file'>  
   specifies the path to the truststore file.  
   Alias: SSLTRUSTSTORE  
   ODBC driver default: $HOME/.ssl/trust.pem

SSLVALIDATECERTIFICATE=YES | NO  
   indicates whether to validate the certificate of the communication partner.  
   Alias: SSLVALIDATECERTIFICATE  
   ODBC driver default: NO

SSLHOSTNAMEINCERTIFICATE=<'string'>  
   specifies the host-name certificate of the keystore.  
   Alias: SSLHOSTNAMEINCERT, SSLHOSTNAMEINCERTIFICATE

SSLCREATESELFSIGNEDCERTIFICATE=YES | NO  
   specifies whether to create a self-signed certificate if the keystore cannot be found.  
   Alias: SSLCREATECERT, SSLCREATESELFSIGNEDCERTIFICATE

LIBNAME-options  
define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to SAP HANA with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

Table 30.1  SAS/ACCESS LIBNAME Options for SAP HANA

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>NO</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DBCOMMIT</strong></td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td><strong>DBCONINIT</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>DBCONTERM</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>DBCREATE_TABLE_OPTS</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>DBGEN_NAME</strong></td>
<td>DBMS</td>
</tr>
<tr>
<td><strong>DBINDEX</strong></td>
<td>NO</td>
</tr>
<tr>
<td><strong>DBLIBINIT</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>DBLIBTERM</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>DBMAX_TEXT</strong></td>
<td>1024</td>
</tr>
<tr>
<td><strong>DBMSTEMP</strong></td>
<td>NO</td>
</tr>
<tr>
<td><strong>DBNULLKEYS</strong></td>
<td>YES</td>
</tr>
<tr>
<td><strong>DBPROMPT</strong></td>
<td>NO</td>
</tr>
<tr>
<td><strong>DBSLICEPARM</strong></td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td><strong>DEFER</strong></td>
<td>NO</td>
</tr>
<tr>
<td><strong>DELETE_MULT_ROWS</strong></td>
<td>NO</td>
</tr>
<tr>
<td><strong>DIRECT_EXE</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>DIRECT_SQL</strong></td>
<td>YES</td>
</tr>
<tr>
<td><strong>IGNORE_READ_ONLY_COLUMNS</strong></td>
<td>NO</td>
</tr>
<tr>
<td><strong>INSERTBUFF</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>LOGIN_TIMEOUT</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>MULTI_DATASRC_OPT</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>PARMDEFAULT</strong></td>
<td>on page 181</td>
</tr>
<tr>
<td><strong>PARMSTRING</strong></td>
<td>on page 181</td>
</tr>
<tr>
<td><strong>PRESERVE_COL_NAMES</strong></td>
<td>NO (see Naming Conventions for SAP HANA) on page 841</td>
</tr>
</tbody>
</table>
### LIBNAME Statement Specifics for SAP HANA

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>NO (see Naming Conventions for SAP HANA) on page 841</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>0</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TABLE_TYPE=</td>
<td>none</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>USE_ODBC_CL=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>

### SAP HANA LIBNAME Statement Examples

In this example, SERVER=, DATABASE=, USER=, and PASSWORD= are connection options. No DSN style is specified. This is the default method, which is recommended.

```
libname A1 saphana server=mysrv1
    port=30015 user=myusr1 password='mypwd1';
```

This example requires that you specify a DSN style.

```
libname B1 saphana dsn=hnatest user=myusr1 password=mypwd1;
```

Here is an example of the LIBNAME statement using the PARMSTRING= option:

```
libname a sasiohna user=userid password=xxxx server=server-name
    instance=02 preserve_tab_names=yes parmrstring="parm_price=30"
    preserve_col_names=yes;
```
All SAS/ACCESS data set options in this table are supported for SAP HANA. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_CONTROL_FIELD_DELIMITER</td>
<td>, (comma)</td>
</tr>
<tr>
<td>BL_CONTROL_RECORD_DELIMITER</td>
<td>&quot;n' (newline)</td>
</tr>
<tr>
<td>BL_CONTROL_QUOTATION_MARK</td>
<td>' (single quotation mark)</td>
</tr>
<tr>
<td>BL_FILE_BADFILE</td>
<td>creates a file in the current directory or with the default file specifications</td>
</tr>
<tr>
<td>BL_FILE_CONTROLFILE</td>
<td>creates a file in the current directory or with the default file specifications</td>
</tr>
<tr>
<td>BL_FILE_DATAFILE</td>
<td>creates a file in the current directory or with the default file specifications</td>
</tr>
<tr>
<td>BL_FILE_DEFAULT_DIR</td>
<td>NO</td>
</tr>
<tr>
<td>BL_FILE_DEFAULT_DIR</td>
<td>'/tmp/'</td>
</tr>
<tr>
<td>BL_IMPORT_BATCH_SIZE</td>
<td>none</td>
</tr>
<tr>
<td>BL_IMPORT_OPTIONS</td>
<td>none</td>
</tr>
<tr>
<td>BL_IMPORT_TABLE_LOCK</td>
<td>NO</td>
</tr>
<tr>
<td>BL_IMPORT_TYPE_CHECK</td>
<td>NO</td>
</tr>
<tr>
<td>BL_SFTP_HOST</td>
<td>none</td>
</tr>
<tr>
<td>BL_SFTP_OPTIONS</td>
<td>none</td>
</tr>
<tr>
<td>BL_SFTP_USER</td>
<td>none</td>
</tr>
<tr>
<td>BL_SFTP_WAIT_MILLISECONDS</td>
<td>none</td>
</tr>
<tr>
<td>BULKLOAD</td>
<td>NO</td>
</tr>
<tr>
<td>DBCOMMIT</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for SAP HANA” on page 842</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for SAP HANA” on page 842</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>PARMDEFAULT= on page 389</td>
<td>none</td>
</tr>
<tr>
<td>PARMSTRING= on page 389</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for SAP HANA

#### Key Information

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the SAP HANA interface.

- The `dbms-name` is **SAPHANA**.
- PROC SQL supports multiple connections to SAP HANA. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default **SAPHANA** alias is used.
- The CONNECT statement `database-connection-arguments` are identical to its LIBNAME connection options.

#### CONNECT Statement Examples

This example connects to SAP HANA and then disconnects from it.

```sql
proc sql noerrorstop;
   connect to saphana as x1(server=mysrv1
      port=30015 user=mysurl password='mypwd1');
   disconnect from x1;
   quit;
```

This next example connects to SAP HANA, executes some SQL statements, and then disconnects from SAP HANA.

```sql
proc sql noerrorstop;
   connect to saphana as x1(server=mysrv1
      port=30015 user=mysurl password='mypwd1'
      execute (CREATE TABLE t1 (no int primary key, state varchar(10))) by x1;
   execute (INSERT INTO t1 values (1, 'USA')) by x1;
   execute (INSERT INTO t1 values (2, 'CHN')) by x1;
   select * from connection to x1 (SELECT * FROM t1 ORDER BY no);
   disconnect from x1;
   quit;
```
Understanding SAP HANA Update and Delete Rules

To avoid data integrity problems when updating or deleting data, you need to define a primary key on your table.

This example uses DBTYPE= to create the primary key.

```
libname invty saphana server=mysrv1 port=30015 user=myusr1;

proc sql;
drop table invty.STOCK23;
quit;

data invty.STOCK23(DBTYPE=(RECDATE="date not null,primary key(RECDATE)");
input PARTNO $ DESCX $ INSTOCK @17 RECDATE date7. @25 PRICE;
format RECDATE date7.;
datalines;
K89R  seal     34  27 Jul 95  245.00
M447  sander   98  20 Jun 95   45.88
LK43  filter  121  19 May 96   10.99
MN21  brace    43  10 Aug 96   27.87
BC85  clamp    80  16 Aug 96    9.55
KJ66  cutter    6  20 Mar 96   24.50
UYN7  rod     211  18 Jun 96   19.77
JD03  switch  383  09 Jan 97   13.99
BVII  timer    26  03 Jan 97   34.50
;
```

These next examples show how you can update the table now that STOCK23 has a primary key.

```
proc sql;
update invty.STOCK23 set price=price*1.1 where INSTOCK > 50;
quit;

proc sql;
delete from invty.STOCK23 where INSTOCK > 150;
quit;
```
Autopartitioning Scheme for SAP HANA

Overview

Autopartitioning for SAS/ACCESS Interface to SAP HANA is a modulo (MOD) function method. For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

Autopartitioning Restrictions

SAS/ACCESS Interface to SAP HANA places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here is how columns are partitioned.

- INTEGER, SMALLINT, and TINYINT columns are given preference.
- You can use DECIMAL, DOUBLE, FLOAT, NUMERIC, and REAL columns for partitioning. These types are converted to BIGINT by using the CAST function.

Nullable Columns

If you select a nullable column for autopartitioning, the OR<column-name>IS NULL SQL statement is appended at the end of the SQL code that is generated for the threaded Read. This ensures that any possible NULL values are returned in the result set.

Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in the WHERE clause. For example, the following DATA step could not use a threaded Read to retrieve the data because all numeric columns in the table are in the WHERE clause.

data work.locemp;
  set hanalib.MYEMPS;
  where EMPNUM<=30 and ISTENURE=0
       and SALARY<=35000 and NUMCLASS>2;
run;

Using DBSLICEPARM=

SAS/ACCESS Interface to SAP HANA defaults to three threads when you use autopartitioning, but do not specify a maximum number of threads in DBSLICEPARM= to use for the threaded Read.

Using DBSLICE=

You might achieve the best possible performance when using threaded Reads by specifying the DBSLICE= option for SAP HANA in your SAS operation. This is especially true if your table uses multiple database partitions. You can also use the
DBSLICE= clause to specify the WHERE clause for each partition, as shown in this example.

```
proc print data=trilib.MYEMPS(DBSLICE=("EMPNUM BETWEEN 1 AND 33"
   "EMPNUM BETWEEN 34 AND 66" "EMPNUM BETWEEN 67 AND 100"));
run;
```

The methods and examples described in DBSLICE= work well in cases where the table that you want to read is not stored in multiple partitions in your DBMS. These methods also give you flexibility in column selection. For example, if you know that the STATE column in your employee table only contains a few distinct values, you can customize your DBSLICE= clause accordingly.

```
datawork.locemp;
set hanalib2.MYEMP(DBSLICE=("STATE='FL'" "STATE='GA'"
   "STATE='SC'" "STATE='VA'" "STATE='NC'");
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

---

**SAP HANA Schema Flexibility**

The SAS/ACCESS engine for SAP HANA limits the number of columns in a table or a view to 64,000. Regular SAP HANA tables and views have a limit of 1,000 columns. If you try to create tables with more columns, the SAP HANA ODBC driver returns an error.

SAP HANA tables with schema flexibility can have up to 64,000 columns. To create SAP HANA tables with schema flexibility, use the following LIBNAME option or data set option:

```
DBCREATE_TABLE_OPTS='WITH SCHEMA FLEXIBILITY'
```

---

**Passing SAS Functions to SAP HANA**

SAS/ACCESS Interface to SAP HANA passes the following SAS functions to SAP HANA for processing. Where the SAP HANA function name differs from the SAS function name, the SAP HANA name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

- **(POWER(base, exponent))**
- ABS
- ARCOS (ACOS)
- ARSIN (ASIN)
- ATAN
- ATAN2
- AVG
- BYTE (CHAR(CAST(expression AS INTEGER)))
- CEIL
- COALESCE
- LOG2 (LOG(2, value))
- LOWCASE (LOWER)
- MAX
- MIN
- MINUTE
- MOD (see note)
- MONTH
- QTR (CAST(TO_CHAR(value, 'Q') AS SMALLINT))
- SECOND
- SIGN
### SQL_FUNCTIONS=ALL

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to SAP HANA. Due to incompatibility in date and time functions between SAP HANA and SAS, SAP HANA might not process them correctly. Check your results to determine whether these functions are working as expected.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>DBMS Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESS (REPLACE)</td>
<td>TIME (CURRENT_TIME)</td>
</tr>
<tr>
<td>DATE (CURRENT_DATE)</td>
<td>TIMEPART (CAST(expression AS TIME))</td>
</tr>
<tr>
<td>DATEPART (CAST(expression AS DATE))</td>
<td>TODAY (CURRENT_DATE)</td>
</tr>
<tr>
<td>DATETIME (CURRENT_TIMESTAMP)</td>
<td>TRANSLATE</td>
</tr>
<tr>
<td>SYSDATE</td>
<td></td>
</tr>
</tbody>
</table>

### Passing Joins to SAP HANA

For a multiple libref join to pass to SAP HANA, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- server (SERVER=)
- instance (INSTANCE=)
- port (PORT=)
- SQL functions (SQL_FUNCTIONS=)
For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for SAP HANA

Overview

Bulk loading is the fastest way to insert large numbers of rows into an SAP HANA table. To use the bulk-load facility, specify BULKLOAD=YES. The bulk-load facility uses the SAP HANA IMPORT statement and Secure Shell (SSH) File Transfer Protocol (FTP), or SFTP, to move data from the client to the SAP HANA database.

Data Set Options for Bulk Loading

Here are the SAP HANA bulk-load data set options. For detailed information about these options, see Data Set Options for Relational Databases on page 241.

- BL_FILE_BADFILE=
- BL_CONTROL_FIELD_DELIMITER=
- BL_CONTROL_RECORD_DELIMITER=
- BL_CONTROL_QUOTATION_MARK=
- BL_FILE_CONTROLFILE=
- BL_FILE_DATAFILE= data set option,
- BL_FILE_DEFAULT_DIR=
- BL_FILE_DEFAULT_DIR=
- BL_IMPORT_BATCH_SIZE=
- BL_IMPORT_OPTIONS=
- BL_IMPORT_TABLE_LOCK=
- BL_IMPORT_TYPE_CHECK=
- BL_SFTP_HOST=
- BL_SFTP_OPTIONS=
- BL_SFTP_USER=
- BL_SFTP_WAIT_MILLISECONDS=
- BULKLOAD=

Naming Conventions for SAP HANA

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The SAP HANA interface supports table names and column names that contain up to 32 characters. If DBMS column names
are longer than 32 characters, they are truncated to 32 characters. If truncating a column name would result in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

The PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options determine how SAS/ACCESS Interface to SAP HANA handles case sensitivity. Although SAP HANA is not case-sensitive, all names are stored in uppercase if you do not enclose them in double quotation marks. These options also control whether you can use an SAP HANA reserved word as an identifier. For more information about these options, see LIBNAME Statement for Relational Databases on page 96.

SAP HANA objects include tables, views, synonyms, columns, indexes, functions, procedures, users, roles, and more. To represent names for these objects in the SQL statement, SAP HANA uses delimited identifiers and undelimited identifiers. An undelimited identifier represents an undelimited table or column name.

Follow these naming conventions:

- An identifier must be from 1 to 127 characters long.
- An undelimited name must begin with a letter (A through Z). It cannot contain any symbols except digits or an underscore (_).
- A delimited name must be enclosed within the double quotation ("”) delimiter. It can contain any character, including special characters (for example, "AB$%CD").
- Names are not case sensitive. For example, CUSTOMER and Customer are the same, but object names are converted to uppercase when they are stored in the SAP HANA database. However, if you enclose a name in quotation marks, it is case sensitive.
- A name cannot be an SAP HANA reserved word.

Note: If PRESERVE_COL_NAMES= is enabled (set to YES), then a column name can be a reserved word. Similarly, if PRESERVE_TAB_NAMES= is enabled (set to YES), then a table name can be a reserved word. By default, these options are disabled.

- A name cannot be the same as another SAP HANA object that has the same type.

For more information, see your SAP HANA documentation.

---

**Data Types for SAP HANA**

**Overview**

Every column in a table has a name and a data type. The data type tells SAP HANA how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about SAP HANA data types, null and default values, and data conversions.

**Data Types for SAP HANA**

**Overview**

Here are the data types that the SAP HANA engine supports.
• Character data:
  VARCHAR(n)
  NCHAR(n)
  ALPHANUM
  SHORTTEXT
• Binary data: VARBINARY
• Large object (LOB) data:
  BLOB (binary large object)
  CLOB (character large object)
  NCLOB
  TEXT
• Numeric data:
  TINYINT
  SMALLINT
  INTEGER
  BIGINT
  REAL
  DOUBLE
  FLOAT(n)
  DECIMAL(precision, scale) or DEC(p, s)

  Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

• Date, time, and timestamp data:
  DATE
  TIME
  SECONDDATE
  TIMESTAMP

**SAP HANA Null Values**

Many relational database management systems have a special value called NULL. A DBMS NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DBMS NULL value, it interprets it as a SAS missing value.

In most relational databases, columns can be defined as NOT NULL so that they require data (they cannot contain NULL values). When a column is defined as NOT NULL, the DBMS does not add a row to the table unless the row has a value for that column. When creating a DBMS table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.
To control how the DBMS handles SAS missing character values, use the `NULLCHAR=` and `NULLCHARVAL=` data set options.

### Geospatial Data

SAS/ACCESS Interface to SAP HANA can import and export geospatial data of the type `ST_GEOMETRY`. For example, you can export the following subtypes of geospatial data to SAP HANA tables:

**ST_POINT**

represents a single location (point) that is represented by a pair of latitude and longitude coordinates. Data of the geospatial type point in an SAP HANA table might look like the following example:

```plaintext
SHAPE.ST_ASGEOJSON()
{"type": "Point", "coordinates": [84.24823999999996, 18.79219399999999]}
```

In a SAS table, the latitude and longitude coordinates are each stored in a numeric variable.

**ST_POLYGON**

represents a set of points that define the perimeter of an area. The points define the vertices of a closed polygon. A record for an item of type `ST_POLYGON` is made of multiple pairs of coordinates that define the points (vertices) in the polygon. The first and last points must contain the same values. Geospatial data of type polygon in an SAP HANA table might look like the following example:

```plaintext
SHAPE.ST_ASWKT()
POLYGON (84.24824 18.79219, 84.25500 18.78855, 84.25984 18.79453, 84.25984 18.78489, 84.28588 18.78654, 84.29696 18.79190, 84.32205 18.79154, 84.33023 18.80136, 84.33841 18.80701, 84.35423 18.84336, 84.35228 18.85309, 84.34839 18.86280, 84.34835 18.87626, 84.35803 18.86846, ...
83.93397 18.64837, 83.93369 18.68782, 83.94396 18.69438, 83.96128 18.67464, 83.99588 18.64173, 84.01654 18.63514, 84.04403 18.63511, 84.05769 18.64825, 84.10570 18.66136, 84.11241 18.68436, 84.11224 18.70737, 84.11905 18.71723, 84.18427 18.72703, 84.19458 18.72702, 84.19444 18.74674, 84.22868 18.76315, 84.24824 18.79219 )
```

This data would be read from multiple records in a SAS data set into a single record in the SAP HANA table. Each row in the SAS data set contains the coordinates from one point (vertex) of a polygon. Multiple rows from the SAS data set are combined and exported into SAP HANA into a variable of data type `ST_POLYGON`.

Other SAP HANA `ST_GEOMETRY` subtypes, in addition to the examples shown here, can be supported.

### Working with Geospatial Data in SAS

#### Overview of Importing Geospatial Data into SAS

In SAP HANA, geospatial data (data of type `ST_GEOMETRY`) can take a number of subtypes, including point (`ST_POINT`), polygon (`ST_POLYGON`), and multi-polygon (`ST_MULTIPLATFORM`). Geospatial data consists of pairs of latitude and longitude coordinate values. These coordinates can be read into numeric variables, such as X and Y, that respectively store the latitude and longitude values for each point. Typically, you must convert relevant parts of geospatial data input into numeric or character data in SAS. If multiple geospatial points (representing longitude and latitude coordinates) are
imported, such as for the subtype ST_POLYGON, the best practice is to import each into separate rows. You can combine the rows before exporting the data back to SAP HANA.

To import geospatial data, import the values by using PROC SQL. Alternatively, you can import SHP files by using PROC MAPIMPORT. For more information, see SAS SQL Procedure User’s Guide, SAS/GRAPH: Reference, and your SAP HANA documentation.

**Example 1: Export Point Data into SAP HANA**

The following code shows how to export point coordinates into a table in SAP HANA:

```sql
execute (insert into "MY_GEOSPATIAL_FROM_SAS"(segment, area, flag, party, pc_code, pc_hname, pc_name, pc_no, pc_type, st_code, st_name, shape)
    (select segment, area, flag, party, pc_code, pc_hname, pc_name, pc_no, pc_type, st_code, st_name,
     new ST_POINT('POINT(' || x || ' ' || y || ')')
     from #temp_a))
by sasiohna;
```

In this example, the numeric variables X and Y in the SAS data set define the latitude and longitude coordinates for a point. The data from table SAS data set TEMP_A is exported into an SAP HANA table MY_GEOSPATIAL_FROM_SAS that contains geospatial data of type ST_POINT.

**Example 2: Export Polygon Data into SAP HANA**

In SAS, data for a polygon that represents a city is stored in multiple rows. Each row contains the latitude and longitude coordinates for a point (vertex) of the polygon that defines the city’s perimeter. All of the rows for one city must be combined into a single row in the target SAP HANA table. Add the coordinates to a single instance of a variable of type ST_POLYGON.

Before exporting coordinates for geospatial data of type ST_POLYGON, generate an intermediate SAS data set that combines the coordinate data from each row into a single record for each polygon. You can do this using FIRST. and LAST. processing and concatenation functions.

In this example, the coordinates for each vertex of the polygon were originally stored in separate variables for the latitude and longitude. These pairs of values are combined into a variable, ST_POLY, that lists the pairs of coordinates separated by commas. A row in the intermediate data set might contain data that looks similar to this:

```
ST_POLY = 84.248239999999996  18.792190000000002, 84.228679999999997 18.763150000000000, 84.194440000000000 18.746739999999999
```

City = Cary
Zip = 27513
State = NC

The following code shows how to export polygon data from the intermediate SAS data set into a table in SAP HANA:

```sql
execute (insert into "MY_GEOSP_POLYGON_FROM_SAS"(segment, area, flag, party, pc_code, pc_hname, pc_name, pc_no, pc_type, st_code, st_name, shape)
    (select segment, area, flag, party, pc_code, pc_hname, pc_name, pc_no, pc_type, st_code, st_name,
     new ST_POLYGON('POLYGON((' || to_char(ST_POLY) || '))')
     from #temp_a))
by sasiohna;
```
In this example, the data in the ST_POLY variable in the intermediate SAS data set TEMP_A is exported into an SAP HANA table called MY_GEOESP_POLYGON_FROM_SAS. This table contains geospatial data of type ST_POLYGON.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to SAP HANA assigns to SAS variables when using the **LIBNAME statement** to read from an SAP HANA table. These default formats are based on SAP HANA column attributes.

**Table 30.2 LIBNAME Statement: Default SAS Formats for SAP HANA Data Types**

<table>
<thead>
<tr>
<th>SAP HANA Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR((n))*</td>
<td>character</td>
<td>$w$</td>
</tr>
<tr>
<td>NVARCHAR((n))*</td>
<td>character</td>
<td>$w$</td>
</tr>
<tr>
<td>ALPHANUM((n))*</td>
<td>character</td>
<td>$w$</td>
</tr>
<tr>
<td>SHORTTEXT((n))*</td>
<td>character</td>
<td>$w$</td>
</tr>
<tr>
<td>VARBINARY((n))*</td>
<td>character</td>
<td>$w$. (where (w) is (2^n))</td>
</tr>
<tr>
<td>BLOB</td>
<td>character</td>
<td>$w$. (where (w) is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>CLOB***</td>
<td>character</td>
<td>$w$. (where (w) is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>NCLOB</td>
<td>character</td>
<td>$w$. (where (w) is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>TEXT</td>
<td>character</td>
<td>$w$. (where (w) is the minimum of 32767 and the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>TINYINT</td>
<td>numeric</td>
<td>4.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>numeric</td>
<td>11.</td>
</tr>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>DECIMAL((\text{precision, scale})) or DEC((p,s))**</td>
<td>numeric</td>
<td>(w.d)</td>
</tr>
<tr>
<td>SMALLDECIMAL</td>
<td>numeric</td>
<td>(w.d)</td>
</tr>
<tr>
<td>SAP HANA Data Type</td>
<td>SAS Data Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT(n)</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME8.</td>
</tr>
<tr>
<td>SECONDDATE</td>
<td>numeric</td>
<td>DATETIME20.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME26.7</td>
</tr>
<tr>
<td>ST GEOMETRY</td>
<td>numeric</td>
<td>none</td>
</tr>
</tbody>
</table>

* \(n\) in SAP HANA character data types is equivalent to \(w\) in SAS formats.  
** \(p\) and \(s\) in SAP HANA numeric data types are equivalent to \(w\) and \(d\) in SAS formats.  
*** The value of the DBMAX_TEXT= option can override these values.
Chapter 31
SAS/ACCESS Interface to Sybase

Introduction to SAS/ACCESS Interface to Sybase ........................................ 850
LIBNAME Statement Specifics for Sybase .................................................. 850
  Overview ......................................................... 850
  Arguments ...................................................... 850
  Sybase LIBNAME Statement Example ................................................. 853
Data Set Options for Sybase ................................................................. 854
SQL Pass-Through Facility Specifics for Sybase ....................................... 855
  Key Information .................................................. 855
  Example ......................................................... 856
Autopartitioning Scheme for Sybase ......................................................... 856
  Overview ......................................................... 856
  Indexes .......................................................... 856
  Partitioning Criteria .............................................. 857
  Data Types ...................................................... 857
  Examples ......................................................... 857
Temporary Table Support for Sybase ......................................................... 858
ACCESS Procedure Specifics for Sybase .................................................. 858
  Overview ......................................................... 858
  Example ......................................................... 858
DBLOAD Procedure Specifics for Sybase .................................................. 859
  Overview ......................................................... 859
  Example ......................................................... 860
Passing SAS Functions to Sybase ............................................................. 861
Passing Joins to Sybase ............................................................................. 862
Bulk Loading for Sybase .......................................................................... 862
  Overview ......................................................... 862
  Data Set Options for Bulk Loading ................................................... 862
Reading Multiple Sybase Tables ............................................................... 862
Locking in the Sybase Interface ............................................................... 863
  Overview ......................................................... 863
  Understanding Sybase Update Rules .................................................. 864
Naming Conventions for Sybase ............................................................... 864
Case Sensitivity in Sybase ........................................................................ 865
### Introduction to SAS/ACCESS Interface to Sybase

For available SAS/ACCESS features, see Sybase supported features on page 88. For more information about Sybase, see your Sybase documentation.

For information about Sybase IQ, see Chapter 32, “SAS/ACCESS Interface to Sybase IQ,” on page 873.

### LIBNAME Statement Specifics for Sybase

#### Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Sybase supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Sybase.

```
LIBNAME libref sybase <connection-options> <LIBNAME-options>;
```

#### Arguments

- **libref**
  - any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

- **sybase**
  - the SAS/ACCESS engine name for the Sybase interface.

- **connection-options**
  - provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. Here are the connection options for Sybase. All options are all case sensitive: They are passed to Sybase exactly as you enter them.

- **USER='SYBASE-user-name'**
  - specifies the Sybase user name (also called the login name) that you use to connect to your database. If the user name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.
PASSWORD=<"SYBASE-password"/>

specifies the password that is associated with the Sybase user name. If you omit the password, a default password of NULL is used. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: PASS=, PW=, SYBPW=

DATABASE=<"Sybase-database-name"/>

specifies the name of the Sybase database that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you omit DATABASE=, the default database for your Sybase user name is used.

Alias: DB=

SERVER=<"server-name"/>

specifies the server that you want to connect to. This server accesses the database that contains the tables and views that you want to access. If the server name contains lowercase, spaces, or nonalphanumeric characters, you must enclose it in quotation marks. If you omit SERVER=, the default action for your operating system occurs. On UNIX systems, the value of the environment variable DSQUERY is used if it has been set.

INTERFACE=filename

specifies the name and location of the Sybase interfaces file. This file contains the names and network addresses of all available servers on the network. If you omit this statement, the default action for your operating system occurs. INTERFACE= is not used in some operating environments. Contact your database administrator to determine whether it applies to your operating environment.

IP_CURSOR= YES | NO

specifies whether implicit PROC SQL pass-through processes multiple result sets simultaneously. IP_CURSOR is set to NO by default. Setting it to YES allows this type of extended processing. However, it decreases performance because cursors, not result sets, are being used. Do not set to YES unless needed.

SYBBUFSZ=number-of-rows

specifies the number of rows of DBMS data to write to the buffer. If this statement is used, the SAS/ACCESS interface view engine creates a buffer that is large enough to hold the specified number of rows. This buffer is created when the associated database table is read. The interface view engine uses SYBBUFSZ= to improve performance. If you omit this statement, no data is written to the buffer.

If you specify the appropriate system options or environment variables for your database, you can often omit the connection options. See your Sybase documentation for details.

**LIBNAME-options**

define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Sybase, with the applicable default values. For details, see **LIBNAME Options for Relational Databases on page 101.**
Table 31.1  SAS/ACCESS LIBNAME Options for Sybase

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTODOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>YES</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>UNIQUE when data source supports only one cursor per connection; otherwise, SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBLINK=</td>
<td>the local database</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSERVER_MAX_BYTES=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS, 2 or 3</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>ENABLE_BULK=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>INTERFACE=</td>
<td>none</td>
</tr>
<tr>
<td>MAX_CONNECTS=</td>
<td>25</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>PACKETSIZE=</td>
<td>server setting</td>
</tr>
<tr>
<td>QUOTED_IDENTIFIER=</td>
<td>NO</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>100</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>1 (see “Locking in the Sybase Interface”)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>NOLOCK (see “Locking in the Sybase Interface”)</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_OJ_ANSI=</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>1 (see “Locking in the Sybase Interface”)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>PAGE (see “Locking in the Sybase Interface”)</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Sybase LIBNAME Statement Example**

In the following example, the libref MYDBLIB uses the Sybase engine to connect to a Sybase database. USER= and PASSWORD= are connection options.

```
libname mydblib sybase user=myusr1 password=mypwd1;
```

If you specify the appropriate system options or environment variables for your database, you can often omit the connection options. See your Sybase documentation for details.
# Data Set Options for Sybase

All SAS/ACCESS data set options in this table are supported for Sybase. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>BULK_BUFFER=</td>
<td>100</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBLINK=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>_ALL_YES</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for Sybase” on page 866</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>DATETIME22.3</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SEGMENT_NAME=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>

**SQL Pass-Through Facility Specifics for Sybase**

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Sybase interface.

- The *dbms-name* is **SYBASE**.
- The CONNECT statement is optional. If you omit the CONNECT statement, an implicit connection is made using the default values for all connection options.
- The interface can connect multiple times to one or more servers.
- The *database-connection-arguments* for the CONNECT statement are identical to its LIBNAME connection options.
- These LIBNAME options are also available with the CONNECT statement.
Example

This example retrieves a subset of rows from the Sybase INVOICE table. Because the WHERE clause is specified in the DBMS query (the inner SELECT statement), the DBMS processes the WHERE expression and returns a subset of rows to SAS.

```sql
proc sql;
connect to sybase(server=MYSRV1 database=INVENTORY
   user=myusr1 password=mypwd1);
%put &sqlxmsg;
select * from connection to sybase
   (select * from INVOICE where BILLEDBY=457232);
%put &sqlxmsg;
```

The SELECT statement that is enclosed in parentheses is sent directly to the database and therefore must be specified using valid database variable names and syntax.

Autopartitioning Scheme for Sybase

Overview

For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

Sybase autopartitioning uses the Sybase MOD function (%) to create multiple SELECT statements with WHERE clauses. In the optimum scenario, the WHERE clauses divide the result set into equal chunks: one chunk per thread. For example, assume that your original SQL statement was `SELECT * FROM DBTAB`, and assume that DBTAB has a primary key column PKCOL of type integer and that you want it partitioned into three threads. Here is how the autopartitioning scheme would break up the table into three SQL statements:

```sql
select * from DBTAB where (abs(PKCOL))%3=0
select * from DBTAB where (abs(PKCOL))%3=1
select * from DBTAB where (abs(PKCOL))%3=2
```

Because PKCOL is a primary key column, you should receive a fairly even distribution among the three partitions, which is the primary goal.

Indexes

An index on a SAS partitioning column increases performance of the threaded Read. If a primary key is not defined for the table, an index should be placed on the partitioning column in order to attain similar benefits. To achieve optimum database performance, it is essential to understand and follow the recommendations in the *Sybase ASE*
Performance and Tuning Guide for creating and using indexes. Here is the order of column selection for the partitioning column.

1. Identity column
2. Primary key column (INTEGER or NUMERIC)
3. INTEGER, NUMERIC, or BIT; not nullable
4. INTEGER, NUMERIC, or BIT; nullable

If the column selected is a bit type, only two partitions are created because the only values are 0 and 1.

**Partitioning Criteria**

The most efficient partitioning column is an Identity column, which is usually identified as a primary key column. Identity columns usually lead to evenly partitioned result sets because of the sequential values that they store.

The least efficient partitioning column is a numeric, decimal, or float column that is NULLABLE, and does not have a defined index.

Given equivalent selection criteria, columns defined at the beginning of the table definition that meet the selection criteria takes precedence over columns defined toward the end of the table definition.

**Data Types**

These data types are supported in partitioning column selection:

- BIT
- DECIMAL
- FLOAT
- INTEGER
- NUMERIC
- SMALLINT
- TINYINT

**Examples**

Here are examples of generated SELECT statements involving various column data types.

COL1 is NUMERIC, DECIMAL, or FLOAT. This example uses three threads (the default) and COL1 is NOT NULL.

```
select * from DBTAB where (abs(convert(INTEGER, COL1)))%3=0
select * from DBTAB where (abs(convert(INTEGER, COL1)))%3=1
select * from DBTAB where (abs(convert(INTEGER, COL1)))%3=2
```

COL1 is BIT, INTEGER, SMALLINT, or TINYINT. This example uses two threads (the default) and COL1 is NOT NULL.

```
select * from DBTAB where (abs(COL1))%3=0
select * from DBTAB where (abs(COL1))%3=1
```
Temporary Table Support for Sybase

SAS/ACCESS Interface to Sybase supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

ACCESS Procedure Specifics for Sybase

Overview

For general information about this feature, see the Appendix 1, “ACCESS Procedure,” on page 955.

SAS/ACCESS for Sybase supports all ACCESS procedure statements. Here are the ACCESS Procedure specifics for Sybase.

- The DBMS= value for PROC ACCESS is SYBASE.
- The database-description-statements that PROC ACCESS uses are identical to the database-connection-arguments in the CONNECT statement for the SQL pass-through facility.
- Here is the TABLE= statement for PROC ACCESS.

\[
\text{TABLE=} \langle\text{table-name}\rangle;
\]

specifies the name of the Sybase table or Sybase view on which the access descriptor is based.

Example

The following example creates access descriptors and view descriptors for the EMPLOYEES and INVOICE tables. These tables have different owners and are stored in PERSONNEL and INVENTORY databases that reside on different machines. The USER= and PASSWORD= statements identify the owners of the Sybase tables and their passwords.

\[
\text{libname \textit{vlib} 'sas-library'; }
\]

\[
\text{proc access dbms=sybase; }
\]

\[
\text{create work.employee.access; }
\]

\[
\text{server='mysrv1'; database='personnel'; }
\]

\[
\text{user='myusr1'; password='mypwd1'; }
\]

\[
\text{table=EMPLOYEES; }
\]

\[
\text{create vlib.emp_acc.view; }
\]

\[
\text{select all; }
\]

\[
\text{format empid 6.; }
\]

\[
\text{subset where DEPT like 'ACC%'; }
\]

\[
\text{run; }
\]
proc access dbms=sybase;
    create work.invoice.access;
    server='mysrv2'; database='inventory';
    user='myusr2'; password='mypwd2';
    table=INVOICE;
    rename invoicenum=invnum;
    format invoicenum 6. billedon date9.
                      paidon date9.;
    create vlib.sainv.view;
    select all;
    subset where COUNTRY in ('Argentina','Brazil');
    run;

options linesize=120;
title 'South American Invoices and
     Who Submitted Them';

proc sql;
    select invnum, country, billedon, paidon,
        billedby, lastname, firstnam
    from vlib.emp_acc, vlib.sainv
    where emp_acc.empid=sainv.billedby;

Sybase is a case-sensitive database. The PROC ACCESS database identification
statements and the Sybase column names in all statements except SUBSET are
converted to uppercase unless the names are enclosed in quotation marks. The SUBSET
statements are passed to Sybase exactly as you enter them, so you must use the correct
case for the Sybase column names.

---

DBLOAD Procedure Specifics for Sybase

**Overview**

For general information about this feature, see the Appendix 3, “DBLOAD Procedure,”
on page 985.

The Sybase interface supports all DBLOAD procedure statements.

Here are the Sybase interface specifics for the DBLOAD procedure.

- The DBMS= value for PROC DBLOAD is **SYBASE**.
- The TABLE= statement for PROC DBLOAD is:
  ```
  TABLE= <>'table-name'<>
  ```
- PROC DBLOAD uses these `database-description-statements`.
  ```
  USER=<>'SYBASE-user-name'<>
  ```
  specifies the Sybase user name (also called the login name) that you use to
  connect to your database. If the user name contains spaces or nonalphanumeric
  characters, you must enclose it in quotation marks.

  ```
  PASSWORD=<>'SYBASE-password'<>
  ```
  specifies the password that is associated with the Sybase user name.
If you omit the password, a default password of NULL is used. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

**PASSWORD=** can also be specified with the **SYBPW=**, **PASS=**, and **PW=** aliases.

**DATABASE=<<database-name>>**

specifies the name of the Sybase database that contains the tables and views that you want to access.

If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you omit **DATABASE=**, the default database for your Sybase user name is used.

Alias: DB=

**SERVER=<<server-name>>**

specifies the server that you want to connect to. This server accesses the database that contains the tables and views that you want to access.

If the server name contains lowercase, spaces, or nonalphanumeric characters, you must enclose it in quotation marks.

If you omit **SERVER=**, the default action for your operating system occurs. On UNIX systems, the value of the environment variable DSQUERY is used if it has been set.

**INTERFACE=filename**

specifies the name and location of the Sybase interfaces file. The interfaces file contains the names and network addresses of all available servers on the network.

If you omit this statement, the default action for your operating system occurs. **INTERFACE=** is not used in some operating environments. Contact your database administrator to determine whether it applies to your operating environment.

**BULKCOPY= Y|N;**

uses the Sybase bulk copy utility to insert rows into a Sybase table. The default value is **N**.

If you specify **BULKCOPY=Y**, **BULKCOPY=** calls the Sybase bulk copy utility in order to load data into a Sybase table. This utility groups rows so that they are inserted as a unit into the new table. Using the bulk copy utility can improve performance.

You use the **COMMIT=** statement to specify the number of rows in each group (this argument must be a positive integer). After each group of rows is inserted, the rows are permanently saved in the table. As each group is being inserted, if one row in the group is rejected, all rows in that group are rejected.

If you specify **BULKCOPY=N**, rows are inserted into the new table using Transact-SQL INSERT statements. See your Sybase documentation for more information about the bulk copy utility.

**Example**

The following example creates a new Sybase table, **EXCHANGE**, from the DLIB.RATEOFEX data file. (The DLIB.RATEOFEX data set is included in the sample data that is shipped with your software.) An access descriptor **ADLIB.EXCHANGE** is also created, and it is based on the new table. The DBLOAD procedure sends a Transact-
SQL GRANT statement to Sybase. You must be granted Sybase privileges to create new Sybase tables or to grant privileges to other users.

libname adlib 'SAS-library';
libname dlib 'SAS-library';

proc dbload dbms=sybase data=dlib.rateofex;
server='mysrv1'; database='testdb'; user='myusr1';
password='mypwd1'; table=EXCHANGE; accdesc=adlib.exchange;
rename fgnindol=fgnindolar 4=dolrsinfgn;
nulls updated=n fgnindol=n 4=n country=n;
load;
run;

---

**Passing SAS Functions to Sybase**

SAS/ACCESS Interface to Sybase passes the following SAS functions to Sybase for processing if the DBMS driver or client that you are using supports the function. Where the Sybase function name differs from the SAS function name, the Sybase name appears in parentheses. For information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>Sybase Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>MAX</td>
</tr>
<tr>
<td>ARCOS (ACOS)</td>
<td>MIN</td>
</tr>
<tr>
<td>ARSIN (ASIN)</td>
<td>MINUTE</td>
</tr>
<tr>
<td>ATAN</td>
<td>MONTH</td>
</tr>
<tr>
<td>AVG</td>
<td>SECOND</td>
</tr>
<tr>
<td>CEIL (CEILING)</td>
<td>SIG</td>
</tr>
<tr>
<td>COS</td>
<td>SIN</td>
</tr>
<tr>
<td>COUNT</td>
<td>SQRT</td>
</tr>
<tr>
<td>DATETIME (GETDATE())</td>
<td>STRIP (RTRIM(LTRIM))</td>
</tr>
<tr>
<td>DAY</td>
<td>SUM</td>
</tr>
<tr>
<td>EXP</td>
<td>TAN</td>
</tr>
<tr>
<td>FLOOR</td>
<td>TRIMN (RTRIM)</td>
</tr>
<tr>
<td>HOUR</td>
<td>UPCASE (UPPER)</td>
</tr>
<tr>
<td>LOG</td>
<td>WEEKDAY</td>
</tr>
<tr>
<td>LOWCASE (LOWER)</td>
<td>YEAR</td>
</tr>
</tbody>
</table>

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Sybase. Due to incompatibility in date and time functions between Sybase and SAS, Sybase might not process them correctly. Check your results to determine whether these functions are working as expected.

DATEPART   TIMEPART
ROUND
Passing Joins to Sybase

For a multiple libref join to pass to Sybase, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- database (DATABASE=)
- server (SERVER=)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for Sybase

Overview

Bulk loading is the fastest way to insert large numbers of rows into a Sybase table. To use the bulk-load facility, specify BULKLOAD=YES. The bulk-load facility uses the Sybase bulk copy facility to move data from SAS to Sybase. See the ENABLE_BULK= LIBNAME option on page 159.

When BULKLOAD=NO, insertions are processed and rolled back as expected according to DBCOMMIT= and ERRLIMIT= values. If the ERRLIMIT= value is encountered, all uncommitted rows are rolled back. The DBCOMMIT= data set option determines the commit intervals.

When BULKLOAD=YES, the first error encountered causes the remaining rows—including the erroneous row—in the buffer to be rejected. No other errors within the same buffer are detected, even if the ERRLIMIT= value is greater than one. In addition, all rows before the error are committed, even if DBCOMMIT= is larger than the number of the erroneous row.

Data Set Options for Bulk Loading

Here are the Sybase bulk-load data set options. For detailed information about these options, see Chapter 12, “Data Set Options for Relational Databases,” on page 237.

- BULK_BUFFER=
- BULKLOAD=

Reading Multiple Sybase Tables

SAS opens multiple Sybase tables for simultaneous reading in these situations:

- When you are using PROC COMPARE. Here is an example:
proc compare base=syb.data1 compare=syb.data2;

- When you are running an SCL program that reads from more than one Sybase table simultaneously.
- When you are joining Sybase tables in SAS—namely, when implicit pass-through is not used (DIRECT_SQL=NO). Here are four examples:

  proc sql;
  select * from syb.table1, syb.table2 where table1.x=table2.x;
  
  proc sql;
  select * from syb.table1 where table1.x = (select x from syb.table2
  where y = 33);
  
  proc sql;
  select empname from syb.employee where empyears > all (select empyears
  from syb.employee where emptitle = 'salesrep');
  
  proc sql;
  create view myview as
  select * from employee where empyears > all (select empyears from
  syb.employee where emptitle = 'salesrep');
  proc print data=myview;

To read two or more Sybase tables simultaneously, you must specify either the LIBNAME option CONNECTION=UNIQUE or the LIBNAME option READLOCK_TYPE=PAGE. Because READLOCK_TYPE=PAGE can degrade performance, it is generally recommended that you use CONNECTION=UNIQUE (unless there is a concern about the number of connections that are opened on the database).

---

Locking in the Sybase Interface

**Overview**

These LIBNAME and data set options let you control how the Sybase interface handles locking. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

READ_LOCK_TYPE= PAGE | NOLOCK

The default value for Sybase is NOLOCK.

UPDATE_LOCK_TYPE= PAGE | NOLOCK

- PAGE
  SAS/ACCESS uses a cursor that you can update. PAGE is the default value for Sybase. When you use this setting, you cannot use the SCHEMA= option, and it is also recommended that the table have a defined primary key.

- NOLOCK
  SAS/ACCESS uses Sybase browse mode updating, in which the table that is being updated must have a primary key and timestamp.

READ_ISOLATION_LEVEL= 1 | 2 | 3

For reads, Sybase supports isolation levels 1, 2, and 3, as defined in the following table. See your Sybase documentation for more information.
Table 31.3 Isolation Levels for Sybase

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prevents dirty Reads. This is the default transaction isolation level.</td>
</tr>
<tr>
<td>2</td>
<td>Uses serialized Reads.</td>
</tr>
<tr>
<td>3</td>
<td>Also uses serialized Reads.</td>
</tr>
</tbody>
</table>

UPDATE_ISOLATION_LEVEL= 1 | 3

Sybase uses a shared or update lock on base table pages that contain rows representing a current cursor position. This option applies to updates only when UPDATE_LOCK_TYPE=PAGE because cursor updating is in effect. It does not apply when UPDATE_LOCK_TYPE=NOLOCK.

For updates, Sybase supports isolation levels 1 and 3, as defined in the preceding table. See your Sybase documentation for more information.

Understanding Sybase Update Rules

To avoid data integrity problems when updating and deleting data in Sybase tables, take these precautionary measures:

- Always define a primary key.
- If the updates are not taking place through cursor processing, define a timestamp column.

It is not always obvious whether updates are using cursor processing. Cursor processing is never used for LIBNAME statement updates if UPDATE_LOCK_TYPE=NOLOCK. Cursor processing is always used in these situations:

- Updates using the LIBNAME statement with UPDATE_LOCK_TYPE=PAGE. This is the default setting for this option.
- Updates using PROC SQL views.
- Updates using PROC ACCESS view descriptors.

Naming Conventions for Sybase

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Sybase database objects include tables, views, columns, indexes, and database procedures. They follow these naming conventions.

- A name must be from 1 to 30 characters long—or 28 characters, if you enclose the name in quotation marks.
- A name must begin with an alphabetic character (A to Z) or an underscore (_) unless you enclose the name in quotation marks.
• After the first character, a name can contain letters (A to Z) in uppercase or lowercase, numbers from 0 to 9, underscore (_), dollar sign ($), number sign (#), at sign (@), yen sign (¥), and monetary pound sign (£).

• Embedded spaces are not allowed unless you enclose the name in quotation marks.

• Embedded quotation marks are not allowed.

• Case sensitivity is set when a server is installed. By default, the names of database objects are case sensitive. For example, the names CUSTOMER and customer are different on a case-sensitive server.

• By default, Sybase does not enclose column names and table names in quotations marks. To enclose these in quotation marks, you must use the QUOTED_IDENTIFIER= LIBNAME option when you assign a libref.

• When you use the DATASETS procedure to list your Sybase tables, the table names appear exactly as they exist in the Sybase data dictionary. If you specified the SCHEMA= LIBNAME option, SAS/ACCESS lists the tables for the specified schema user name.

• To reference a table or other named object that you own, or for the specified schema, use the table name (for example, CUSTOMERS). If you use the DBLINK= LIBNAME option, all references to the libref refer to the specified database.

• A name cannot be a reserved word in Sybase unless the name is enclosed in quotation marks. See your Sybase documentation for more information about reserved words.

• Database names must be unique. For each owner within a database, names of database objects must be unique. Column names and index names must be unique within a table.

---

Case Sensitivity in Sybase

SAS names can be entered in either uppercase or lowercase. When you reference Sybase objects through the SAS/ACCESS interface, objects are case sensitive and require no quotation marks.

However, Sybase is generally set for case sensitivity. Give special consideration to the names of such objects as tables and columns when the SAS ACCESS or DBLOAD procedures are to use them. The ACCESS procedure converts Sybase object names to uppercase unless they are enclosed in quotation marks. Any Sybase objects that were given lowercase names, or whose names contain national or special characters, must be enclosed in quotation marks. The only exceptions are the SUBSET statement in the ACCESS procedure and the SQL statement in the DBLOAD procedure. Arguments or values from these statements are passed to Sybase exactly as you enter them, with the case preserved.

In the SQL pass-through facility, all Sybase object names are case sensitive. The names are passed to Sybase exactly as they are entered.

For more information about case sensitivity and Sybase names, see “Naming Conventions for Sybase” on page 864.
Data Types for Sybase

Overview

Every column in a table has a name and a data type. The data type indicates to the DBMS how much physical storage to reserve for the column and the format in which the data is stored. This section includes information about Sybase data types, null values, and data conversions, and also explains how to insert text into Sybase from SAS.

SAS/ACCESS does not support these Sybase data types: BINARY, VARBINARY, IMAGE, NCHAR(n), and NVARCHAR(n). SAS/ACCESS provides an error message when it tries to read a table that has at least one column that uses an unsupported data type.

Supported Sybase Data Types

Here are the supported Sybase data types for SAS/ACCESS Interface to Sybase:

- Character data:
  
  CHAR(n)       TEXT
  VARCHAR(n)

  Note: You must enclose all character data in single or double quotation marks.

- Numeric data:

  Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

  NUMERIC(p,s)  TINYINT
  DECIMAL(p,s)  SMALLINT
  REAL          INT
  FLOAT         BIT

- Date, time, and money data:

  DATE          TIMESTAMP
  TIME          SMALLMONEY
  SMALLDATETIME MONEY
  DATETIME

866  Chapter 31  •  SAS/ACCESS Interface to Sybase
**User-Defined Data**

You can supplement the Sybase system data types by defining your own data types with the Sybase system procedure `sp_addtype`. When you define your own data type for a column, you can specify a default value (other than NULL) for it and define a range of allowable values for it.

**Sybase Null Values**

Sybase has a special value that is called NULL. A NULL value indicates an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Sybase NULL value, it interprets it as a SAS missing value.

By default, Sybase columns are defined as NOT NULL. NOT NULL tells Sybase not to add a row to the table unless the row has a value for the specified column.

If you want a column to accept NULL values, you must explicitly define it as NULL. Here is an example of a CREATE TABLE statement that defines all table columns as NULL except CUSTOMER. In this case, Sybase accepts a row only if it contains a value for CUSTOMER.

```sql
create table CUSTOMERS
   (CUSTOMER        char(8)    not null,
    STATE           char(2)        null,
    ZIPCODE         char(5)        null,
    COUNTRY         char(20)       null,
    TELEPHONE       char(12)       null,
    NAME            char(60)       null,
    CONTACT         char(30)       null,
    STREETADDRESS   char(40)       null,
    CITY            char(25)       null,
    FIRSTORDERDATE  datetime       null);
```

When you create a Sybase table with SAS/ACCESS, you can use the `DBNULL=` data set option to indicate whether NULL is a valid value for specified columns.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how SAS missing character values are handled, use the `NULLCHAR=` and `NULLCHARVAL=` data set options.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to Sybase assigns to SAS variables when using the `LIBNAME` statement to read from a Sybase table. These default formats are based on Sybase column attributes.

<table>
<thead>
<tr>
<th>Sybase Column Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)</td>
<td>character</td>
<td>$n^*$</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>character</td>
<td>$n^*$</td>
</tr>
<tr>
<td>Sybase Column Type</td>
<td>SAS Data Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>TEXT</td>
<td>character</td>
<td>$n$** (where $n$ is the value of the DBMAX_TEXT= option)</td>
</tr>
<tr>
<td>BIT</td>
<td>numeric</td>
<td>1.0</td>
</tr>
<tr>
<td>TINYINT</td>
<td>numeric</td>
<td>4.0</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.0</td>
</tr>
<tr>
<td>INT</td>
<td>numeric</td>
<td>11.0</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>numeric</td>
<td>$w, w.d$ (if possible)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>numeric</td>
<td>$w, w.d$ (if possible)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>numeric</td>
<td>BEST22.</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>BEST11.</td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>numeric</td>
<td>DOLLAR12.2</td>
</tr>
<tr>
<td>MONEY</td>
<td>numeric</td>
<td>DOLLAR24.2</td>
</tr>
<tr>
<td>DATE*</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIME*</td>
<td>numeric</td>
<td>TIME12.</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>numeric</td>
<td>DATETIME22.3</td>
</tr>
<tr>
<td>DATETIME</td>
<td>numeric</td>
<td>DATETIME22.3</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>hexadecimal</td>
<td>$\text{SHEX}w$</td>
</tr>
</tbody>
</table>

* If a conflict might occur between the Sybase and SAS value for this data type, use SASDATEFMT= to specify the SAS format.
** $n$ specifies the current value for the Adaptive Server page size.

This table shows the default Sybase data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

**Table 31.5 LIBNAME STATEMENT: Default Sybase Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Sybase Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sw$, SCHAR$w$, SVARYING$w$, SHEX$w$.</td>
<td>VARCHAR($w$)</td>
</tr>
<tr>
<td>DOLLAR$w.d$</td>
<td>SMALLMONEY (where $w &lt; 6$)</td>
</tr>
<tr>
<td></td>
<td>MONEY (where $w &gt;= 6$)</td>
</tr>
</tbody>
</table>
### SAS Variable Format

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Sybase Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>datetime format</td>
<td>DATETIME</td>
</tr>
<tr>
<td>date format</td>
<td>DATE</td>
</tr>
<tr>
<td>time format</td>
<td>TIME</td>
</tr>
<tr>
<td>any numeric with a SAS format name of w.d (where d &gt; 0 and w &gt; 10) or w</td>
<td>NUMERIC(p,s)</td>
</tr>
<tr>
<td>any numeric with a SAS format name of w.d (where d = 0 and w &lt; 10)</td>
<td>TINYINT (where w &lt; 3) SMALLINT (where w &lt; 5) INT (where w &lt; 10)</td>
</tr>
<tr>
<td>any other numeric</td>
<td>FLOAT</td>
</tr>
</tbody>
</table>

You can override these default data types by using the `DBTYPE=` on page 351 data set option.

### ACCESS Procedure Data Conversions

This table shows the default SAS variable formats that SAS/ACCESS assigns to Sybase data types when you use the `ACCESS` procedure.

<table>
<thead>
<tr>
<th>Sybase Column Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)</td>
<td>character</td>
<td>$n. (n &lt;= 200)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$200. (n &gt; 200)</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>character</td>
<td>$n. (n &lt;= 200)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$200. (n &gt; 200)</td>
</tr>
<tr>
<td>BIT</td>
<td>numeric</td>
<td>1.0</td>
</tr>
<tr>
<td>TINYINT</td>
<td>numeric</td>
<td>4.0</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.0</td>
</tr>
<tr>
<td>INT</td>
<td>numeric</td>
<td>11.0</td>
</tr>
<tr>
<td>FLOAT</td>
<td>numeric</td>
<td>BEST22.</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>BEST11.</td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>numeric</td>
<td>DOLLAR12.2</td>
</tr>
</tbody>
</table>
Sybase Column Type | SAS Data Type | Default SAS Format
--- | --- | ---
MONEY | numeric | DOLLAR24.2
SMALLDATETIME | numeric | DATETIME21.2
DATETIME | numeric | DATETIME21.2

The ACCESS procedure also supports Sybase user-defined data types. The ACCESS procedure uses the Sybase data type on which a user-defined data type is based in order to assign a default SAS format for columns.

The DECIMAL, NUMERIC, and TEXT data types are not supported in PROC ACCESS. The TIMESTAMP data type is not displayed in PROC ACCESS.

**DBLOAD Procedure Data Conversions**

This table shows the default Sybase data types that SAS/ACCESS assigns to SAS variable formats when you use the **DBLOAD procedure on page 985**.

**Table 31.7** PROC **DBLOAD: Default Sybase Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Sybase Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w., $CHAR$w., $VARYING$w., $HEX$w.</td>
<td>CHAR($w$)</td>
</tr>
<tr>
<td>$w.$</td>
<td>TINYINT</td>
</tr>
<tr>
<td>$w.$</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>$w.$</td>
<td>INT</td>
</tr>
<tr>
<td>$w.$</td>
<td>FLOAT</td>
</tr>
<tr>
<td>$w.d$</td>
<td>FLOAT</td>
</tr>
<tr>
<td>$IB$w.d, $PIB$w.d</td>
<td>INT</td>
</tr>
<tr>
<td>FRACT, E format, and other numeric formats</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DOLLAR$w.d$, $w$$&lt;=12$</td>
<td>SMALLMONEY</td>
</tr>
<tr>
<td>DOLLAR$w.d$, $w$$&gt;12$</td>
<td>MONEY</td>
</tr>
<tr>
<td>any datetime, date, or time format</td>
<td>DATETIME</td>
</tr>
</tbody>
</table>

The DBLOAD procedure also supports Sybase user-defined data types. Use the **TYPE=** statement to specify a user-defined data type.
Data Returned as SAS Binary Data with Default Format $HEX

- BINARY
- VARBINARY
- IMAGE

Data Returned as SAS Character Data

- NCHAR
- NVARCHAR

Inserting TEXT into Sybase from SAS

You can insert only TEXT data into a Sybase table by using the BULKLOAD= data set option, as in this example:

data yourlib.newtable(bulkload=yes);
  set work.sasbigtext;
run;

If you do not use the BULKLOAD= option, you receive this error message:

ERROR: Object not found in database. Error Code: -2782
An untyped variable in the PREPARE statement 'S401bcf78'
is being resolved to a TEXT or IMAGE type.
This is illegal in a dynamic PREPARE statement.

National Language Support for Sybase

To support output and update processing from SAS into Sybase in languages other than English, special setup steps are required. These setup steps are required so that date, time, and datetime values can be processed correctly. In SAS, you must ensure that the DFLANG= system option is set to the correct language. A system administrator can set this globally or a user can set it within a single SAS session. In Sybase, the default client language, set in the locales.dat file, must match the language that is used in SAS.
Chapter 32
SAS/ACCESS Interface to Sybase IQ

Introduction to SAS/ACCESS Interface to Sybase IQ ........................................... 874
LIBNAME Statement Specifics for Sybase IQ ...................................................... 874
  Overview ................................................. 874
  Arguments .............................................. 874
  Sybase IQ LIBNAME Statement Example ...................................................... 877
Data Set Options for Sybase IQ ................................................................. 878
SQL Pass-Through Facility Specifics for Sybase IQ ........................................ 880
  Key Information ......................................... 880
  CONNECT Statement Example .............................................................. 880
  Special Catalog Queries ................................................................. 880
Autopartitioning Scheme for Sybase IQ ......................................................... 881
  Overview .................................................. 881
  Autopartitioning Restrictions .......................................................... 881
  Nullable Columns .............................................. 882
  Using WHERE Clauses ................................................. 882
  Using DBSLICE=.............................................. 882
  Using DBSLICEPARM= ........................................ 882
Passing SAS Functions to Sybase IQ .......................................................... 883
Passing Joins to Sybase IQ ........................................................................ 884
Bulk Loading for Sybase IQ ................................................................. 884
  Loading ...................................................... 884
  Data Set Options for Bulk Loading ...................................................... 884
  Examples ..................................................... 885
Locking in the Sybase IQ Interface ............................................................. 885
Naming Conventions for Sybase IQ ............................................................ 887
Data Types for Sybase IQ ................................................................. 887
  Overview ...................................................... 887
  Supported Sybase IQ Data Types ...................................................... 888
  Sybase IQ Null Values ......................................................... 888
  LIBNAME Statement Data Conversions ................................................ 889
Introduction to SAS/ACCESS Interface to Sybase IQ

For available SAS/ACCESS features, see Sybase IQ supported features on page 89. For more information about Sybase IQ, see your Sybase IQ documentation.

For information about Sybase, see Chapter 31, “SAS/ACCESS Interface to Sybase,” on page 849.

LIBNAME Statement Specifics for Sybase IQ

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Sybase IQ supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Sybase IQ.

```
LIBNAME libref sybaseiq <connection-options> <LIBNAME-options>;
```

Arguments

`libref`

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

`sybaseiq`

specifies the SAS/ACCESS engine name for the SybaseIQ interface.

`connection-options`

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to the Sybase IQ database in two ways. Specify only one of these methods for each connection because they are mutually exclusive.

- `HOST=, SERVER=, DATABASE=, PORT=, USER=, PASSWORD=`
- `DSN=, USER=, PASSWORD=`

Here is how these options are defined.

`HOST=<>'server-name'>'`

specifies the host name or IP address where the Sybase IQ database is running. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

`SERVER=<>'server-name'>'`

specifies the Sybase IQ server name, also known as the engine name. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.
DATABASE= database-name
specifies the Sybase IQ database that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: DB=

PORT= port
specifies the port number that is used to connect to the specified Sybase IQ database. If you do not specify a port, the default is 2638.

USER= Sybase-IQ-user-name
specifies the Sybase IQ user name (also called the user ID) that you use to connect to your database. If the user name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

PASSWORD= Sybase-IQ-password
specifies the password that is associated with your Sybase IQ user name. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. You can also specify PASSWORD= with the PWD=, PASS=, and PW= aliases.

DSN= Sybase-IQ-data-source
specifies the configured Sybase IQ ODBC data source to which you want to connect. Use this option if you have existing Sybase IQ ODBC data sources that are configured on your client. This method requires additional setup—either through the ODBC Administrator control panel on Windows platforms or through the odbc.ini file or a similarly named configuration file on UNIX platforms. So it is recommended that you use this connection method only if you have existing, functioning data sources that have been defined.

LIBNAME-options
define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Sybase IQ, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

Table 32.1 SAS/ACCESS LIBNAME Options for Sybase IQ

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>operation-specific</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td></td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>YES</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DELETE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>LOGIN_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>see “Naming Conventions for Sybase IQ”</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>see “Naming Conventions for Sybase IQ”</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>0</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the Sybase IQ Interface”)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>automatically calculated based on row length</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the Sybase IQ Interface”)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Sybase IQ LIBNAME Statement Example**

In this example, HOST=, SERVER=, DATABASE=, USER=, and PASSWORD= are connection options.

```plaintext
libname mydblib sybaseiq host=iqsvr1 server=iqsrv1_users db=users user=iqusr1 password=iqpwd1;

proc print data=mydblib.customers;
    where state='CA';
run;
```

In the next example, DSN=, USER=, and PASSWORD= are connection options. The SybaseIQ SQL data source is configured in the ODBC Administrator Control Panel on Windows platforms or in the odbc.ini file or a similarly named configuration file on UNIX platforms.

```plaintext
libname mydblib sybaseiq DSN=SybaseIQSQL user=iqusr1 password=iqpwd1;
```
Data Set Options for Sybase IQ

All SAS/ACCESS data set options in this table are supported for Sybase IQ. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

Table 32.2  SAS/ACCESS Data Set Options for Sybase IQ

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_CLIENT_DATAFILE=</td>
<td>none</td>
</tr>
<tr>
<td>BL_DATAFILE=</td>
<td>When BL_USE_PIPE=NO, creates a file in the current directory or with the default file specifications.</td>
</tr>
<tr>
<td>BL_DELETE_DATAFILE=</td>
<td>YES (only when BL_USE_PIPE=NO)</td>
</tr>
<tr>
<td>BL_DELIMITER=</td>
<td></td>
</tr>
<tr>
<td>BL_OPTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>BL_SERVER_DATAFILE=</td>
<td>creates a data file in the current directory or with the default file specifications (same as for BL_DATAFILE=)</td>
</tr>
<tr>
<td>BL_USE_PIPE=</td>
<td>YES</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see “Data Types for Sybase IQ”</td>
</tr>
<tr>
<td>DBSLICE=</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Data Types for Sybase IQ”</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>IGNORE_READ_ONLY_COLUMNS=</td>
<td>NO</td>
</tr>
<tr>
<td>INSERTBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>
SQL Pass-Through Facility Specifics for Sybase IQ

Key Information

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Sybase IQ interface.

• The `dbms-name` is `SYBASEIQ`.

• The CONNECT statement is required.

• PROC SQL supports multiple connections to Sybase IQ. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default `sybaseiq` alias is used.

• The CONNECT statement `database-connection-arguments` are identical to its `LIBNAME connection-options`.

CONNECT Statement Example

This example uses the DBCON alias to connection to the `iqsvr1` Sybase IQ database and execute a query. The connection alias is optional.

```sql
proc sql;
  connect to sybaseiq as dbcon
      (host=iqsvr1 server=iqsvr1_users db=users user=iqusr1 password=iqpwd1);
  select * from connection to dbcon
      (select * from customers where customer like '1%');
quit;
```

Special Catalog Queries

SAS/ACCESS Interface to Sybase IQ supports the following special queries. You can the queries use to call the ODBC-style catalog function application programming interfaces (APIs). Here is the general format of the special queries:

`SIQ::SQLAPI "parameter 1","parameter n"`

`SIQ::` is required to distinguish special queries from regular queries. SIQ:: is not case sensitive.

`SQLAPI` is the specific API that is being called. SQLAPI is not case sensitive.

"parameter n"

a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign matches any sequence of zero or more characters, and the underscore represents any single character. To use either character as a literal value, you can use the backslash character (\) to escape the match characters. For
example, this call to SQLTables usually matches table names such as myatest and my_test:

```sql
select * from connection to sybaseiq (SIQ::SQLTables "test","","my_test");
```

Use the escape character to search only for the my_test table:

```sql
select * from connection to sybaseiq (SIQ::SQLTables "test","","my\_test");
```

SAS/ACCESS Interface to Sybase IQ supports these special queries.

- **SIQ::SQLTables <"Catalog", "Schema", "Table-name", "Type">**
  returns a list of all tables that match the specified arguments. If you do not specify any arguments, all accessible table names and information are returned.

- **SIQ::SQLColumns <"Catalog", "Schema", "Table-name", "Column-name">**
  returns a list of all columns that match the specified arguments. If you do not specify any argument, all accessible column names and information are returned.

- **SIQ::SQLPrimaryKeys <"Catalog", "Schema", "Table-name">**
  returns a list of all columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If you do not specify any table name, this special query fails.

- **SIQ::SQLSpecialColumns <"Identifier-type", "Catalog-name", "Schema-name", "Table-name", "Scope", "Nullable">**
  returns a list of the optimal set of columns that uniquely identify a row in the specified table.

- **SIQ::SQLStatistics <"Catalog", "Schema", "Table-name">**
  returns a list of the statistics for the specified table name. You can set SQL_INDEX_ALL and SQL_ENSURE options in the SQLStatistics API call. If you do not specify any table name argument, this special query fails.

- **SIQ::SQLGetTypeInfo**
  returns information about the data types that the Sybase IQ database supports.

---

### Autopartitioning Scheme for Sybase IQ

#### Overview

Autopartitioning for SAS/ACCESS Interface to Sybase IQ is a modulo (MOD) function method. For general information about this feature, see “Autopartitioning Techniques in SAS/ACCESS” on page 59.

#### Autopartitioning Restrictions

SAS/ACCESS Interface to Sybase IQ places additional restrictions on the columns that you can use for the partitioning column during the autopartitioning phase. Here is how columns are partitioned.

- **INTEGER, SMALLINT, and TINYINT columns are given preference.**

- **You can use DECIMAL, DOUBLE, FLOAT, NUMERIC, or NUMERIC columns for partitioning if the precision minus the scale of the column is greater than 0 but less than 10—namely, 0<(precision-scale)<10.**
Nullable Columns

If you select a nullable column for autopartitioning, the `OR<column-name>IS NULL` SQL statement is appended at the end of the SQL code that is generated for the threaded Read. This ensures that any possible NULL values are returned in the result set. Also, if the column to be used for partitioning is defined as BIT, the number of threads are automatically changed to two, regardless how DBSLICEPARM= is set.

Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in a SAS WHERE clause. For example, this DATA step cannot use a threaded Read to retrieve the data because all numeric columns in the table are in the WHERE clause:

```sas
data work.locemp;
set iqlib.MYEMPS;
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

Using DBSLICEPARM=

Although SAS/ACCESS Interface to Sybase IQ defaults to three threads when you use autopartitioning, do not specify a maximum number of threads for the threaded Read in the `DBSLICEPARM= LIBNAME` option on page 151.

Using DBSLICE=

You might achieve the best possible performance when using threaded Reads by specifying the `DBSLICE=` data set option for Sybase IQ in your SAS operation. This is especially true if you defined an index on one of the columns in the table. SAS/ACCESS Interface to Sybase IQ selects only the first integer-type column in the table. This column might not be the same column where the index is defined. If so, you can specify the indexed column using `DBSLICE=`, as shown in this example.

```sas
proc print data=iqlib.MYEMPS(DBSLICE=('EMPNUM BETWEEN 1 AND 33' 'EMPNUM BETWEEN 34 AND 66' 'EMPNUM BETWEEN 67 AND 100'));
run;
```

Using `DBSLICE=` also gives you flexibility in column selection. For example, if you know that the STATE column in your employee table contains only a few distinct values, you can customize your `DBSLICE=` clause accordingly.

```sas
datawork.locemp;
set iqlib2.MYEMP(DBSLICE=('STATE='FL' 'STATE='GA' 'STATE='SC' 'STATE='VA' 'STATE='NC'));
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```
SAS/ACCESS Interface to Sybase IQ passes the following SAS functions to Sybase IQ for processing. Where the Sybase IQ function name differs from the SAS function name, the Sybase IQ name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

ABS
ACOS (ACOS)
ACSC (ASIN)
ATAN
AVG
BYTE (CHAR)
CEIL
COALESCE
COS
COUNT
DAY
EXP
FLOOR
HOUR
INDEX (LOCATE)
LOG
LOG10
LOWER
MAX
MIN
MINUTE
MONTH
QTR (QUARTER)
REPEAT
SECOND
SIGN
SIN
SQRT
STRIP (TRIM)
SUBSTR (SUBSTRING)
SUM
TAN
TRANWRD (REPLACE)
TRIMN (RTRIM)
UPCASE (UPPER)
WEEKDAY (DOW)
YEAR

Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Sybase IQ. Due to incompatibility in date and time functions between Sybase IQ and SAS, Sybase IQ might not process them correctly. Check your results to determine whether these functions are working as expected.

COMPRESS (REPLACE) SOURDEX
DATE (CURRENT_DATE) TIME (CURRENT_TIME)
DATEPART (DATE) TIMEPART (TIME)
DATETIME (CURRENT_TIMESTAMP) TODAY (CURRENT_DATE)
LENGTH (BYTE_LENGTH) TRIM
Passing Joins to Sybase IQ

For a multiple libref join to pass to Sybase IQ, all of these components of the LIBNAME statements must match exactly.

- user ID (USER=)
- password (PASSWORD=)
- host (HOST=)
- server (SERVER=)
- database (DATABASE=)
- port (PORT=)
- data source (DSN=, if specified)
- SQL functions (SQL_FUNCTIONS=)

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

Bulk Loading for Sybase IQ

Loading

Bulk loading is the fastest way to insert large numbers of rows into a Sybase IQ table. You must specify BULKLOAD=YES to use the bulk-load facility. The bulk-load facility uses the Sybase IQ LOAD TABLE command to move data from the client to the Sybase IQ database.

Data Set Options for Bulk Loading

Here are the Sybase IQ bulk-load data set options. For detailed information about these options, see Data Set Options for Relational Databases on page 241.

- BL_CLIENT_DATAFILE=
- BL_DATAFILE=
- BL_DELETE_DATAFILE=
- BL_DELIMITER=
- BL_OPTIONS=
- BL_SERVER_DATAFILE=
- BL_USE_PIPE=
- BULKLOAD=
**Examples**

In this example, the SASFLT.FLT98 SAS data set creates and loads FLIGHTS98, a large Sybase IQ table. For Sybase IQ 12.x, this works only when the Sybase IQ server is on the same server as your SAS session.

```
libname sasflt 'SAS-library';
libname mydblib sybaseiq host=iqsvr1 server=iqsrv1_users
db=users user=iqusr1 password=iqpwd1;
proc sql;
.create table mydblib.flights98
(bulkload=YES)
 as select * from sasflt.flt98;
quit;
```

When the Sybase IQ server and your SAS session are not on the same server, you need to include additional options, as shown in this example.

```
libname sasflt 'SAS-library';
libname mydblib sybaseiq host=iqsvr1 server=iqsrv1_users
db=users user=iqusr1 password=iqpwd1;
proc sql;
create table mydblib.flights98
(  BULKLOAD=YES
  BL_USE_PIPE=NO
  BL_SERVER_DATAFILE='/tmp/fltdata.dat'
  BL_CLIENT_DATAFILE='/tmp/fltdata.dat' )
 as select * from sasflt.flt98;
quit;
```

In this example, you can append the SASFLT.FLT98 SAS data set to the existing Sybase IQ table, ALLFLIGHTS. The BL_USE_PIPE=NO option forces SAS/ACCESS Interface to Sybase IQ to write data to a flat file, as specified in the BL_DATAFILE= option. Rather than deleting the data file, BL_DELETE_DATAFILE=NO causes the engine to leave it after the load has completed.

```
proc append base=mydblib.allflights
(BULKLOAD=YES
  BL_DATAFILE='/tmp/fltdata.dat'
  BL_USE_PIPE=NO
  BL_DELETE_DATAFILE=NO)
data=sasflt.flt98;
run;
```

---

**Locking in the Sybase IQ Interface**

These LIBNAME and data set options let you control how the Sybase IQ interface handles locking. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

- **READ_LOCK_TYPE= ROW | TABLE**
- **UPDATE_LOCK_TYPE= ROW | TABLE**
Sybase IQ supports the S, RR, RC, and RU isolation levels that are defined in this table.

Table 32.3 Isolation Levels for Sybase IQ

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (serializable)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads.</td>
</tr>
<tr>
<td>RR (repeatable read)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RC (read committed)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RU (read uncommitted)</td>
<td>Allows dirty Reads, nonrepeatable Reads, and phantom Reads.</td>
</tr>
</tbody>
</table>

Here are how the terms in the table are defined.

**Dirty reads**
A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it can see changes that are made by those concurrent transactions even before they commit.

For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.

**Nonrepeatable reads**
If a transaction exhibits this phenomenon, it might read a row once and later fail when it attempts to read that row again in the same transaction. The row might have been changed or even deleted by a concurrent transaction. Therefore, the read is not necessarily repeatable.

For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.

**Phantom reads**
When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist, a phantom.

Sybase IQ supports the S, RR, and RC isolation levels defined in the preceding table.
Naming Conventions for Sybase IQ

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. SAS/ACCESS Interface to Sybase IQ supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name would result in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less because SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view. For more information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Sybase IQ is not case sensitive, so all names default to lowercase.

Sybase IQ objects include tables, views, and columns. They follow these naming conventions.

• A name must be from 1 to 128 characters long.
• A name must begin with a letter (A through Z), underscore (_), at sign (@), dollar sign ($), or number sign (#).
• Names are not case sensitive. For example, CUSTOMER and Customer are the same, but object names are converted to lowercase when they are stored in the Sybase IQ database. However, if you enclose a name in quotation marks, it is case sensitive.
• A name cannot be a Sybase IQ reserved word, such as WHERE or VIEW.
• A name cannot be the same as another Sybase IQ object that has the same type.

Data Types for Sybase IQ

Overview

Every column in a table has a name and a data type. The data type tells Sybase IQ how much physical storage to set aside for the column and the form in which the data is stored. This information includes information about Sybase IQ data types, null and default values, and data conversions.

For more information about Sybase IQ data types and to determine which data types are available for your version of Sybase IQ, see your Sybase IQ documentation.

SAS/ACCESS Interface to Sybase IQ does not directly support any data types that are not listed below. Any columns using these types are read into SAS as character strings.
Supported Sybase IQ Data Types

Here are the data types that SAS/ACCESS Interface to Sybase IQ supports:

- **Character data:**
  
  - `CHAR(n)`
  - `VARCHAR(n)`
  - `LONG VARCHAR(n)`

- **Numeric data:**
  
  **Note:** When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

  - `BIGINT`
  - `INTEGER`
  - `BIT`
  - `REAL`
  - `DECIMAL | DEC | NUMERIC`
  - `SMALLINT`
  - `DOUBLE | DOUBLE PRECISION`
  - `TINYINT`
  - `FLOAT`

- **Date, Time, and Timestamp data:**

  **Note:** SQL date and time data types are collectively called datetime values. The SQL data types for dates, times, and timestamps are listed here. Be aware that columns of these data types can contain data values that are out of range for SAS.

  - `DATE`
  - `TIMESTAMP`
  - `TIME`

Sybase IQ Null Values

Sybase IQ has a special value called NULL. A Sybase IQ NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Sybase IQ NULL value, it interprets it as a SAS missing value.

You can define a column in a Sybase IQ table so that it requires data. To do this in SQL, you specify a column as NOT NULL. This tells SQL to allow only a row to be added to a table if a value exists for the field. For example, NOT NULL assigned to the CUSTOMER field in the SASDEMO.CUSTOMER table does not allow a row to be added unless there is a value for CUSTOMER. When creating a table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

You can also define Sybase IQ columns as NOT NULL DEFAULT. For more information about using the NOT NULL DEFAULT value, see your Sybase IQ documentation.

Once you know whether a Sybase IQ column enables NULLs or the host system supplies a default value for a column that is defined as NOT NULL WITH DEFAULT,
you can write selection criteria and enter values to update a table. Unless a column is
defined as NOT NULL or NOT NULL DEFAULT, it allows NULL values.

For more information about how SAS handles NULL values, see “Potential Result Set
Differences When Processing Null Data” on page 33.

To control how SAS missing character values are handled, use the NULLCHAR= and
NULLCHARVAL= data set options.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to Sybase IQ assigns
to SAS variables when using the LIBNAME statement to read from a Sybase IQ table.
These default formats are based on Sybase IQ column attributes.

<table>
<thead>
<tr>
<th>Sybase IQ Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR((n))^</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>VARCHAR((n))^</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>LONG VARCHAR((n))^</td>
<td>character</td>
<td>$w.</td>
</tr>
<tr>
<td>BIGINT</td>
<td>numeric</td>
<td>20.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>numeric</td>
<td>6.</td>
</tr>
<tr>
<td>TINYINT</td>
<td>numeric</td>
<td>4.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>numeric</td>
<td>11.</td>
</tr>
<tr>
<td>BIT</td>
<td>numeric</td>
<td>1.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>REAL</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT</td>
<td>numeric</td>
<td>none</td>
</tr>
<tr>
<td>DECIMAL((p,s))^</td>
<td>numeric</td>
<td>(w.d)</td>
</tr>
<tr>
<td>NUMERIC((p,s))^</td>
<td>numeric</td>
<td>(w.d)</td>
</tr>
<tr>
<td>TIME</td>
<td>numeric</td>
<td>TIME8.</td>
</tr>
<tr>
<td>DATE</td>
<td>numeric</td>
<td>DATE9.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>numeric</td>
<td>DATETIME25.6</td>
</tr>
</tbody>
</table>

* in Sybase IQ character data types is equivalent to \(w\) in SAS formats.
** If a conflict might occur between the Sybase and SAS value for this data type, use SASDATEFMT= to specify the SAS format.
This table shows the default Sybase IQ data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

**Table 32.5 LIBNAME Statement: Default Sybase IQ Data Types for SAS Variable Formats**

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Sybase IQ Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.d</td>
<td>DECIMAL(p,s)**</td>
</tr>
<tr>
<td>other numerics</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>$w</td>
<td>VARCHAR(n)*</td>
</tr>
<tr>
<td>datetime formats</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>date formats</td>
<td>DATE</td>
</tr>
<tr>
<td>time formats</td>
<td>TIME</td>
</tr>
</tbody>
</table>

* n in Sybase IQ data types is equivalent to w in SAS formats.

** p and s in Sybase IQ numeric data types are equivalent to w and d in SAS formats.
Chapter 33
SAS/ACCESS Interface to Teradata

Introduction to SAS/ACCESS Interface to Teradata ........................................... 892
  Overview ........................................................................................................ 892
The SAS/ACCESS Teradata Client ................................................................. 892
LIBNAME Statement Specifics for Teradata ...................................................... 893
  Overview ........................................................................................................ 893
  Arguments ..................................................................................................... 893
Teradata LIBNAME Statement Examples ...................................................... 898
Data Set Options for Teradata ........................................................................ 898
SQL Pass-Through Facility Specifics for Teradata ......................................... 901
  Key Information ............................................................................................ 901
  Examples ....................................................................................................... 902
Temporary Table Support for Teradata .......................................................... 904
Passing SAS Functions to Teradata ................................................................. 904
Passing Joins to Teradata ................................................................................ 905
Maximizing Teradata Load and Read Performance ......................................... 905
  Overview ........................................................................................................ 905
  Using the TPT API ....................................................................................... 905
  Legacy Load and Read Utilities ................................................................... 909
  Using FastLoad ............................................................................................. 910
  Using MultiLoad .......................................................................................... 911
  Examples ....................................................................................................... 913
Autopartitioning Scheme for Teradata (Legacy) ............................................ 914
  Overview ........................................................................................................ 914
  FastExport and Case Sensitivity .................................................................. 915
  FastExport Password Security ..................................................................... 915
  FastExport Setup ......................................................................................... 916
  Using FastExport ......................................................................................... 916
  FastExport and Explicit SQL ....................................................................... 917
  Exceptions to Using FastExport ................................................................... 917
  Threaded Reads with Partitioning WHERE Clauses ....................................... 917
  FastExport versus Partitioning WHERE Clauses .......................................... 918
Teradata Processing Tips for SAS Users ......................................................... 918
  Reading from and Inserting to the Same Teradata Table ............................. 918
  Using a BY Clause to Order Query Results ............................................... 919
  Using TIME and TIMESTAMP .................................................................... 920
  Replacing PROC SORT with a BY Clause .................................................. 921
Introduction to SAS/ACCESS Interface to Teradata

Overview

For available SAS/ACCESS features, see Teradata supported features on page 90. For more information about Teradata, see your Teradata documentation.

Note: SAS/ACCESS Interface to Teradata does not support the DBLOAD and ACCESS procedures. The LIBNAME engine technology enhances and replaces the functionality of these procedures. Therefore, you must revise SAS jobs that were written for a different SAS/ACCESS interface and that include DBLOAD or ACCESS procedures. You need to do this revision before you can run the SAS jobs with SAS/ACCESS Interface to Teradata.

The SAS/ACCESS Teradata Client

Teradata is a massively parallel (MPP) RDBMS. A high-end Teradata server supports many users. It simultaneously loads and extracts table data and processes complex queries. Upsert processing is also supported using MultiLoad.
Because Teradata customers run many processors at the same time for queries of the database, users enjoy excellent DBMS server performance. The challenge to client software, such as SAS, is to leverage Teradata performance by rapidly extracting and loading table data. SAS/ACCESS Interface to Teradata meets this challenge by letting you optimize extracts and loads (reads and creates).

Information throughout this document explains how you can use the SAS/ACCESS interface to optimize DBMS operations.

- It supports the Teradata Parallel Transporter (TPT) API on Windows and UNIX. This API uses the Teradata Load, Update, and Stream operators to load data and the export operator to read data.
- It can create and update Teradata tables. It supports a FastLoad interface that rapidly creates new table. It can also potentially optimize table reads by using FastExport for the highest possible read performance.
- It supports MultiLoad, which loads both empty and existing Teradata tables and greatly accelerates the speed of insertion into Teradata tables.

---

**LIBNAME Statement Specifics for Teradata**

**Overview**

This section describes the LIBNAME statement that SAS/ACCESS Interface to Teradata supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Teradata.

```
LIBNAME libref teradata <connection-options> <LIBNAME-options>;
```

**Arguments**

- **libref**
  - specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

- **teradata**
  - specifies the SAS/ACCESS engine name for the Teradata interface.

- **connection-options**
  - provides connection information and controls how SAS manages the timing and concurrence of the connection to the DBMS. Here are the connection options for the Teradata interface.

  ```
  USER='Teradata-user-name' | ldapid@LDAP | ldapid@LDAPrealm-name
  ```

  - specifies a required connection option that specifies a Teradata user name. If the name contains blanks or national characters, enclose it in quotation marks. For LDAP authentication with either a NULL or single realm, append only the @LDAP token to the Teradata user name. In this case, no realm name is needed. If you append a realm name, the LDAP authentication server ignores it and authentication proceeds. However, if multiple realms exist, you must append the realm name to the @LDAP token. In this case, an LDAP server must already be configured to accept authentication requests from the Teradata server.
PASSWORD=<"Teradata-password">  
specifies a required connection option that specifies a Teradata password. The password that you specify must be correct for your USER= value. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you do not want to enter your Teradata password in clear text on this statement, see PROC PWENCODE in the Base SAS Procedures Guide for a method for encoding it. For LDAP authentication, you use this password option to specify the authentication string or password.

ACCOUNT=<"account_ID">  
is an optional connection option that specifies the account number that you want to charge for the Teradata session.

TDPID=<"dbname">  
Alias: SERVER=

specifies a required connection option if you run more than one Teradata server. TDPID= operates differently for network-attached and channel-attached systems, as described below.

- For NETWORK-ATTACHED systems (PC and UNIX), dbname specifies an entry in your (client) HOSTS file that provides an IP address for a database server connection.

  By default, SAS/ACCESS connects to the Teradata server that corresponds to the dbccop1 entry in your HOSTS file. When you run only one Teradata server and your HOSTS file defines the dbccop1 entry correctly, you do not need to specify TDPID=.

  However, if you run more than one Teradata server, you must use the TDPID= option to specifying a dbname of eight characters or less. SAS/ACCESS adds the specified dbname to the login string that it submits to Teradata. (Teradata documentation refers to this name as the tdpid component of the login string.)

  After SAS/ACCESS submits a dbname to Teradata, Teradata searches your HOSTS file for all entries that begin with the same dbname. For Teradata to recognize the HOSTS file entry, the dbname suffix must be COPx (x is a number). If there is only one entry that matches the dbname, x must be 1. If there are multiple entries for the dbname, x must begin with 1 and increment sequentially for each related entry. (See the example HOSTS file entries below.)

  When there are multiple, matching entries for a dbname in your HOSTS file, Teradata does simple load balancing by selecting one of the Teradata servers that are specified for login. Teradata distributes your queries across these servers so that it can return your results as fast as possible.

  The TDPID= examples below assume that your HOSTS file contains these dbname entries and IP addresses.

- Example 1: The TDPID= option is not specified, establishing a login to the Teradata server that runs at 10.25.20.34: 
  \[\text{dbccop1} 10.25.20.34\]

- Example 2: Using TDPID= myserver or SERVER=myserver, you specify a login to the Teradata server that runs at 130.96.8.207: 
  \[\text{myservercop1 130.96.8.207}\]

- Example 3: Using TDPID=xyz or SERVER=xyz, you specify a login to a Teradata server that runs at 11.22.33.44 or to a Teradata server that runs at 33.44.55.66: 
  \[\text{xyzcop1 33.44.55.66 or xyzcop2 11.22.33.44}\]
To support data parcels that are longer than 64K, the Teradata interface queries the Teradata database to assess Alternate Parcel Header (APH) extended-parcel sizes. In such cases, it needs the Teradata server name (dbcname) as an entry in the HOSTS file or NAMES database to successfully configure the Teradata client for APH processing. The value of the SERVER= (TDPID=) LIBNAME option must be the unqualified short name, not the fully qualified IP name (for example, SERVER=foo rather than SERVER=foo.my.unx.com).

- For CHANNEL-ATTACHED systems (z/OS), TDPID= specifies the subsystem name. This name must be TDPx, where x can be 0–9, A–Z (not case sensitive), or $, # or @. If there is only one Teradata server, and your z/OS system administrator has set up the HSISPB and HSHSPB modules, you do not need to specify TDPID=. For further information, see your Teradata TDPID documentation for z/OS.

DATABASE=<'database-name'><>
   Alias: DB=

   specifies an optional connection option to indicate that the LIBNAME connection be established by specifying the DATABASE= name as the default database. This lets the user directly access another database as the user’s default database, eliminating the need to qualify database objects. The user is actually logged in to the specified database. You must first set permissions to allow user access. If you do not specify DATABASE= in the LIBNAME statement, then the user’s default database is whatever you already previously defined for the user. This usually is a database with the same name as the user name.

SCHEMA=<'database-name'><>
   specifies an optional connection option that you can use to fully qualify objects in another database for SQL processing.

LIBNAME-options define how SAS processes DBMS objects. Some LIBNAME options can enhance performance, and others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Teradata, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>BULKLOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>CAST=</td>
<td>none</td>
</tr>
<tr>
<td>CAST_OVERHEAD_MAXPERCENT</td>
<td>20%</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD for channel-attached systems (z/OS); UNIQUE for network attached systems (UNIX and PC platforms)</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DATABASE= (see SCHEMA= )</td>
<td>none</td>
</tr>
<tr>
<td>DBCLIENT_MAX_BYTES=</td>
<td>1</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (when inserting rows), 0 (when updating rows)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBMSTEMP=</td>
<td>NO</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARM=</td>
<td>THREADED_APPS,2</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td></td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>1 million</td>
</tr>
<tr>
<td>FASTEXPORT=</td>
<td>NO</td>
</tr>
<tr>
<td>LOGDB=</td>
<td>Default Teradata database for the libref</td>
</tr>
<tr>
<td>MODE=</td>
<td>ANSI</td>
</tr>
<tr>
<td>MULTISTMT=</td>
<td>NO</td>
</tr>
<tr>
<td>MULTI_DATASRC_OPT=</td>
<td>IN_CLAUSE</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>YES</td>
</tr>
<tr>
<td>QUERY_BAND=</td>
<td>none</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>see “Locking in the Teradata Interface”</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>none</td>
</tr>
<tr>
<td>READ_MODE_WAIT=</td>
<td>none</td>
</tr>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SAS_DBMS_AUTOMETADATA=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>your default Teradata database</td>
</tr>
<tr>
<td>SESSIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SLEEP=</td>
<td>6</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SQLGENERATION=</td>
<td>DBMS</td>
</tr>
<tr>
<td>TEMPORAL_QUALIFIER=</td>
<td>CURRENT VALIDTIME for valid-time column; CURRENT TRANSACTIONTIME for transaction-time column</td>
</tr>
<tr>
<td>TENACITY=</td>
<td>0 for FastLoad; 4 for FastExport and MultiLoad</td>
</tr>
<tr>
<td>TPT=</td>
<td>YES</td>
</tr>
<tr>
<td>TPT_MAX_SESSIONS=</td>
<td>4</td>
</tr>
<tr>
<td>TPT_MIN_SESSIONS=</td>
<td>1</td>
</tr>
<tr>
<td>TR_ENABLE_INTERRUPT=</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>see “Locking in the Teradata Interface”</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_MODE_WAIT=</td>
<td>none</td>
</tr>
</tbody>
</table>
**Teradata LIBNAME Statement Examples**

These examples show how to make the proper connection by using the USER= and PASSWORD= connection options. Teradata requires these options, and you must use them together.

This example shows how to connect to a single or NULL realm.

```sql
libname x teradata user="johndoe@LDAP" password="johndoeworld";
```

Here is an example of how to make the connection to a specific realm where multiple realms are configured.

```sql
libname x teradata user="johndoe@LDAPjsrealm" password="johndoeworld";
```

---

### Data Set Options for Teradata

All SAS/ACCESS data set options in this table are supported for Teradata. Default values are provided where applicable. For details, see *Data Set Options for Relational Databases* on page 241.

#### Table 33.2 SAS/ACCESS Data Set Options for Teradata

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BL_CONTROL</strong>=</td>
<td>creates a FastExport control file in the current directory with a platform-specific name</td>
</tr>
<tr>
<td><strong>BL_DATAFILE</strong>=</td>
<td>creates a MultiLoad script file in the current directory or with a platform-specific name</td>
</tr>
<tr>
<td><strong>BL_LOG</strong>=</td>
<td>FastLoad errors are logged in Teradata tables named TableName_SE1_randnum and TableName_SE2_randnum. TableName consists of up to the first 15 characters of the target table name, and randnum is a randomly generated number.</td>
</tr>
<tr>
<td><strong>BUFFERS</strong>=</td>
<td>2</td>
</tr>
<tr>
<td><strong>BULKLOAD</strong>=</td>
<td>NO</td>
</tr>
<tr>
<td><strong>CAST</strong>=</td>
<td>none</td>
</tr>
<tr>
<td><strong>CAST_OVERHEAD_MAXPERCENT</strong>=</td>
<td>20%</td>
</tr>
<tr>
<td><strong>DBCOMMIT</strong>=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>DBCONDITION</td>
<td>none</td>
</tr>
<tr>
<td>DBCONSTRAINT</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY</td>
<td>none</td>
</tr>
<tr>
<td>DLBLABEL</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER</td>
<td>none</td>
</tr>
<tr>
<td>DBNULL</td>
<td>none</td>
</tr>
<tr>
<td>DBSASLABEL</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE</td>
<td>see “Data Types for Teradata”</td>
</tr>
<tr>
<td>DBSLICE</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM</td>
<td>THREADED_APPS,2</td>
</tr>
<tr>
<td>DBTYPE</td>
<td>see “Data Types for Teradata”</td>
</tr>
<tr>
<td>ERRLIMIT</td>
<td>1</td>
</tr>
<tr>
<td>FASTEXPORT</td>
<td>NO</td>
</tr>
<tr>
<td>FASTLOAD</td>
<td>NO</td>
</tr>
<tr>
<td>MBUFFSIZE</td>
<td>0</td>
</tr>
<tr>
<td>ML_CHECKPOINT</td>
<td>none</td>
</tr>
<tr>
<td>ML_ERROR1</td>
<td>none</td>
</tr>
<tr>
<td>ML_ERROR2</td>
<td>none</td>
</tr>
<tr>
<td>ML_LOG</td>
<td>none</td>
</tr>
<tr>
<td>ML_RESTART</td>
<td>none</td>
</tr>
<tr>
<td>ML_WORK</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>MULTILOAD=</td>
<td>NO</td>
</tr>
<tr>
<td>MULTISTMT=</td>
<td>NO</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>YES</td>
</tr>
<tr>
<td>QUERY_BAND=</td>
<td>none</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>READ_MODE_WAIT=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>the current LIBNAME option setting</td>
</tr>
<tr>
<td>SESSIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SET=</td>
<td>NO</td>
</tr>
<tr>
<td>SLEEP=</td>
<td>6</td>
</tr>
<tr>
<td>TEMPORAL_QUALIFIER=</td>
<td>CURRENT VALIDTIME [valid-time column] CURRENT TRANSACTIONTIME [transaction-time column]</td>
</tr>
<tr>
<td>TENACITY=</td>
<td>0 [FastLoad], 4 [FastExport, MultiLoad]</td>
</tr>
<tr>
<td>TPT=</td>
<td>YES</td>
</tr>
<tr>
<td>TPT_APPL_PHASE=</td>
<td>NO</td>
</tr>
<tr>
<td>TPT_BUFFER_SIZE=</td>
<td>64</td>
</tr>
<tr>
<td>TPT_CHECKPOINT_DATA=</td>
<td>none</td>
</tr>
</tbody>
</table>
### SQL Pass-Through Facility Specifics for Teradata

#### Key Information

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Teradata interface.
• The dbms-name is TERADATA.
• The CONNECT statement is required.
• The Teradata interface can connect to multiple Teradata servers and to multiple Teradata databases. However, if you use multiple simultaneous connections, you must use an alias argument to identify each connection.
• The CONNECT statement database-connection-arguments are identical to its LIBNAME connection options.

The MODE= LIBNAME option is available with the CONNECT statement. By default, SAS/ACCESS opens Teradata connections in ANSI mode. In contrast, most Teradata tools, such as BTEQ, run in Teradata mode. If you specify MODE=TERADATA, pass-through connections open in Teradata mode, forcing Teradata mode rules for all SQL requests that are passed to the Teradata DBMS. For example, MODE= impacts transaction behavior and case sensitivity. See your Teradata SQL reference documentation for a complete discussion of ANSI versus Teradata mode.
• By default, SAS/ACCESS opens Teradata in ANSI mode. You must therefore use one of these techniques when you write PROC SQL steps that use the SQL pass-through facility.
  • Specify an explicit COMMIT statement to close a transaction. You must also specify an explicit COMMIT statement after any data definition language (DDL) statement. The examples below demonstrate these rules. For further information about ANSI mode and DDL statements, see your Teradata SQL reference documentation.
  • Specify MODE=TERADATA in your CONNECT statement. When MODE=TERADATA, you do not specify explicit COMMIT statements as described above. When MODE=TERADATA, data processing is not case sensitive. This option is available when you use the LIBNAME statement and also with the SQL pass-through facility.

CAUTION:
Do not issue a Teradata DATABASE statement within the EXECUTE statement in PROC SQL. Add the DATABASE= option to your CONNECT statement if you must change the default Teradata database.

CAUTION:
Do not use '--' for comments in PROC SQL. Although Teradata recognizes '--' as a comment, SAS does not. If you use '--' as a comment in PROC SQL, clauses following '--' are considered by Teradata as part of the comment. Use /* */ for comments in PROC SQL.

As a best practice, set the SAS_NON_XVIEW_TABLES environment variable to YES when you invoke SAS. Enabling SAS_NON_XVIEW_TABLES has been shown to improve performance when loading data. Include this code in your SAS command:

```sas
-set SAS_NON_XVIEW_TABLES YES
```

**Examples**

In this example, SAS/ACCESS connects to the Teradata DBMS using the dbcon alias.

```sas
proc sql;
    connect to teradata as dbcon (user=myusr1 pass=mypwd1);
quit;
```
In the next example, SAS/ACCESS connects to the Teradata DBMS using the `tera` alias, drops and re-creates the SALARY table, inserts two rows, and disconnects from the Teradata DBMS. Note that COMMIT must follow each DDL statement. DROP TABLE and CREATE TABLE are DDL statements. The COMMIT statement that follows the INSERT statement is also required. Otherwise, Teradata rolls back the inserted rows.

```
proc sql;
   connect to teradata as tera ( user=myusr1 password=mypwd1 );
   execute (drop table salary) by tera;
   execute (commit) by tera;
   execute (create table salary (current_salary float, name char(10)))
       by tera;
   execute (commit) by tera;
   execute (insert into salary values (35335.00, 'Dan J.')) by tera;
   execute (insert into salary values (40300.00, 'Irma L.')) by tera;
   execute (commit) by tera;
   disconnect from tera;
quit;
```

For this example, SAS/ACCESS connects to the Teradata DBMS using the `tera` alias, updates a row, and disconnects from the Teradata DBMS. The COMMIT statement causes Teradata to commit the update request. Without the COMMIT statement, Teradata rolls back the update.

```
proc sql;
   connect to teradata as tera ( user=myusr1 password=mypwd1 );
   execute (update salary set current_salary=45000
       where (name='Irma L.')) by tera;
   execute (commit) by tera;
   disconnect from tera;
quit;
```

In this example, SAS/ACCESS uses the `tera2` alias to connect to the Teradata database, selects all rows in the SALARY table, displays them using PROC SQL, and disconnects from the Teradata database. No COMMIT statement is needed in this example because the operations are reading only data. No changes are made to the database.

```
proc sql;
   connect to teradata as tera2 ( user=myusr1 password=mypwd1 );
   select * from connection to tera2 (select * from salary);
   disconnect from tera2;
quit;
```

In this next example, MODE=TERADATA is specified to avoid case-insensitive behavior. Because Teradata Mode is used, SQL COMMIT statements are not required.

```
/* Create & populate the table in Teradata mode (case insensitive). */
proc sql;
   connect to teradata (user=myusr1 pass=mypwd1 mode=teradata);
   execute(create table casetest(x char(28)) ) by teradata;
   execute(insert into casetest values('Case Insensitivity Desired') )
       by teradata;
quit;
/* Query the table in Teradata mode (for case-insensitive match). */
proc sql;
   connect to teradata (user=myusr1 pass=mypwd1 mode=teradata);
   select * from connection to teradata (select * from casetest where x='case insensitivity desired');
```
Temporary Table Support for Teradata

SAS/ACCESS Interface to Teradata supports temporary tables. For more information, see “Temporary Table Support for SAS/ACCESS” on page 40.

Passing SAS Functions to Teradata

SAS/ACCESS Interface to Teradata passes the following SAS functions to Teradata for processing. Where the Teradata function name differs from the SAS function name, the Teradata name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>Teradata Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>MAX</td>
</tr>
<tr>
<td>ACOS</td>
<td>MIN</td>
</tr>
<tr>
<td>ARCOSH (ACOSH)</td>
<td>MINUTE</td>
</tr>
<tr>
<td>ARSINH (ASINH)</td>
<td>MOD (see note)</td>
</tr>
<tr>
<td>ASIN</td>
<td>MONTH</td>
</tr>
<tr>
<td>ATAN</td>
<td>SECOND</td>
</tr>
<tr>
<td>ATAN2</td>
<td>SIN</td>
</tr>
<tr>
<td>AVG</td>
<td>SINH</td>
</tr>
<tr>
<td>COALESCE</td>
<td>SQRT</td>
</tr>
<tr>
<td>COS</td>
<td>STD (STDDEV_SAMP)</td>
</tr>
<tr>
<td>COSH</td>
<td>STRIP (TRIM)</td>
</tr>
<tr>
<td>COUNT</td>
<td>SUBSTR</td>
</tr>
<tr>
<td>DATEPART</td>
<td>TAN</td>
</tr>
<tr>
<td>DAY</td>
<td>TANH</td>
</tr>
<tr>
<td>EXP</td>
<td>TIMEPART</td>
</tr>
<tr>
<td>HOUR</td>
<td>TRIM</td>
</tr>
<tr>
<td>INDEX (POSITION)</td>
<td>UPCODE</td>
</tr>
<tr>
<td>LOG</td>
<td>VAR (VAR_SAMP)</td>
</tr>
<tr>
<td>LOG10</td>
<td>YEAR</td>
</tr>
<tr>
<td>LOWCASE (LCASE)</td>
<td></td>
</tr>
</tbody>
</table>

Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see “Functions Where Results Might Vary: MOD Function” on page 44.

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can SAS/ACCESS Interface to Teradata also pass these SAS SQL functions to Teradata. Due to incompatibility in date and time functions between Teradata and SAS, Teradata might not process them correctly. Check your results to determine whether these functions are working as expected. For more information, see SQL_FUNCTIONS=LIBNAME Option on page 213.
DATE
SOUNDEX
DATETIME (current_timestamp)
TIME (current_time)
LEFT (TRIM)
TODAY
LENGTH (CHARACTER_LENGTH)

DATETIME, SOUNDEX, and TIME are not entirely compatible with the corresponding SAS functions. Also, for SOUNDEX, although Teradata always returns 4 characters, SAS might return more or less than 4 characters.

---

Passing Joins to Teradata

For a multiple libref join to pass to Teradata, all of these components of the LIBNAME statements must match exactly:

- user ID (USER=)
- password (PASSWORD=)
- account ID (ACCOUNT=)
- server (TDPID= or SERVER=)

You must specify the SCHEMA= LIBNAME option to fully qualify each table name in a join for each LIBNAME that you reference.

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” on page 45.

---

Maximizing Teradata Load and Read Performance

Overview

SAS/ACCESS supports the Teradata Parallel Transporter application programming interface (TPT API). TPT is now the DEFAULT action for UNIX and PC hosts. FastLoad and MultiLoad are considered to be legacy features on UNIX and PC hosts. For more information, see the Teradata Parallel Transporter documentation.

Note: If for some reason the TPT method is not available, SAS/ACCESS automatically uses FastLoad, FastExport, or MultiLoad methods to insert or retrieve data. By default, you are notified in the SAS log that an alternative method was used. If you want to suppress these messages, set the SAS_TPT_SHOW_MSGS environment variable to NO before you start SAS.

Using the TPT API

TPT API Supported Features and Restrictions

SAS/ACCESS Interface to Teradata supports the TPT API for loading data. The TPT API provides a consistent interface for Fastload, FastExport, MultiLoad, and Multi-Statement insert. TPT API documentation refers to Fastload as the Load operator, FastExport as the Export operator, MultiLoad as the Update operator, and Multi-
Statement insert as the *Stream operator*. SAS supports all these load methods and can restart loading from checkpoints when you use the TPT API with any of them.

**TPT API Setup**

Here are the requirements for using the TPT API in SAS for loading SAS.

- Loading data from SAS to Teradata using the TPT API requires Teradata client TTU 8.2 or later. Verify that you have applied all of the latest Teradata eFixes.
- This feature is supported only on platforms for which Teradata provides the TPT API.
- The native TPT API infrastructure must be present on your system. Contact Teradata if you do not already have it but want to use it with SAS.
- These Teradata privileges are needed to load a Teradata table using the TPT API. For more information, see the Teradata TPT API documentation.
  - CREATE TABLE
  - DROP TABLE
  - CREATE MACRO (Multi-Statement Stream operator)
  - DROP MACRO (Multi-Statement Stream operator)
  - INSERT
  - DELETE
  - SELECT

The SAS configuration document for your system contains information about how to configure SAS to work with the TPT API. However, those steps might already have been completed as part of the post-installation configuration process for your site.

**TPT API LIBNAME Options**

These LIBNAME options are common to all three supported load methods:

- LOGDB=
- SLEEP=
- TENACITY=
- TPT=

If SAS cannot use the TPT API, it reverts to using Fastload, MultiLoad, or Multi-Statement insert, depending on which method of loading was requested without generating any errors.

**TPT API Data Set Options**

These data set options are common to all three supported load methods:

- SLEEP=
- TENACITY=
- TPT=
- TPT_CHECKPOINT_DATA=
- TPT_DATA_ENCRYPTION=
- TPT_LOG_TABLE=
- **TPT_MAX_SESSIONS**
- **TPT_MIN_SESSIONS**
- **TPT_RESTART**
- **TPT_TRACE_LEVEL**
- **TPT_TRACE_LEVEL_INF**
- **TPT_TRACE_OUTPUT**

You can use the `DBCOMMIT= LIBNAME` option and the `CHECKPOINT=` data set option to specify checkpoint frequency with TPT for Fastload, Multiload, and Multi-Statement insert. These options are disabled only for the non-TPT MultiLoad interface and are not relevant with TPT FastExport. Note that SAS sets `TPT_MAX_SESSIONS` to 4.

**TPT API FastLoad Supported Features and Restrictions**

SAS/ACCESS Interface to Teradata supports the TPT API for FastLoad, also known as the *Load operator*. SAS/ACCESS works by interfacing with the Load operator through the TPT API, which in turn uses the Teradata Fastload protocol for loading data. See your Teradata documentation for more information about the Load operator.

This is the default FastLoad method. If SAS cannot find the Teradata modules that are required for the TPT API or if `TPT=NO`, SAS/ACCESS uses the old method of Fastload. SAS/ACCESS can restart Fastload from checkpoints when FastLoad uses the TPT API. The SAS/ACCESS FastLoad facility using the TPT API is similar to the native Teradata FastLoad utility. They share these limitations.

- FastLoad can load only empty tables. It cannot append to a table that already contains data. If you try to use FastLoad when appending to a table that contains rows, the append step fails.
- Data errors are logged in Teradata tables. Error recovery can be difficult if you do not set `TPT_CHECKPOINT_DATA` to enable restart from the last checkpoint. To find the error that corresponds to the code that is stored in the error table, see your Teradata documentation. You can restart a failed job for the last checkpoint by following the instructions in the SAS error log.
- FastLoad does not load duplicate rows (those where all corresponding fields contain identical data) into a Teradata table. If your SAS data set contains duplicate rows, you can use other load methods.

**Starting FastLoad with the TPT API**

See the SAS configuration document for instructions on setting up the environment so that SAS can find the TPT API modules.

You can use one of these options to start FastLoad in the SAS/ACCESS interface using the TPT API:

- the `TPT=YES` data set option in a processing step that populates an empty Teradata table
- the `TPT=YES LIBNAME` option on the destination libref (the Teradata DBMS library where one or more tables are to be created and loaded)

**FastLoad with TPT API Data Set Options**

These data set options are specific to FastLoad using the TPT API:

- **TPT_BUFFER_SIZE**
TPT API MultiLoad Supported Features and Restrictions

SAS/ACCESS Interface to Teradata supports the TPT API for MultiLoad, also known as the Update operator. SAS/ACCESS works by interfacing with the Update operator through the TPT API. This API then uses the Teradata Multiload protocol for loading data. See your Teradata documentation for more information about the Update operator.

This is the default MultiLoad method. If SAS cannot find the Teradata modules that are required for the TPT API or TPT=NO, SAS/ACCESS uses the old method of MultiLoad. SAS/ACCESS can restart Multiload from checkpoints when MultiLoad uses the TPT API.

The SAS/ACCESS MultiLoad facility loads both empty and existing Teradata tables. SAS/ACCESS supports only Insert operations and loading only one target table at a time.

The SAS/ACCESS MultiLoad facility using the TPT API is similar to the native Teradata MultiLoad utility. A common limitation that they share is that you must drop these items onto target tables before the load:

- unique secondary indexes
- foreign key references
- join indexes

Errors are logged to Teradata tables. Error recovery can be difficult if you do not set TPT_CHECKPOINT_DATA= to enable restart from the last checkpoint. To find the error that corresponds to the code that is stored in the error table, see your Teradata documentation. You can restart a failed job for the last checkpoint by following the instructions in the SAS error log.

Starting MultiLoad with the TPT API

See the SAS configuration document for instructions on setting up the environment so that SAS can find the TPT API modules.

You can use one of these options to start MultiLoad in the SAS/ACCESS interface using the TPT API:

- the TPT=YES data set option in a processing step that populates an empty Teradata table
- the TPT=YES LIBNAME option on the destination libref (the Teradata DBMS library where one or more tables are to be created and loaded)

MultiLoad with TPT API LIBNAME Options

The LOGDB= LIBNAME option is specific to MultiLoad using the TPT API.

MultiLoad with TPT API Data Set Options

These data set options are specific to MultiLoad using the TPT API:

- TPT_BUFFER_SIZE=
- TPT_ERROR_TABLE_1=
- TPT_ERROR_TABLE_2=T
**TPT API Multi-Statement Insert Supported Features and Restrictions**

SAS/ACCESS Interface to Teradata supports the TPT API for Multi-Statement insert, also known as the Stream operator. SAS/ACCESS works by interfacing with the Stream operator through the TPT API, which in turn uses the Teradata Multi-Statement insert (TPump) protocol for loading data. See your Teradata documentation for more information about the Stream operator.

This is the default Multi-Statement insert method. If SAS cannot find the Teradata modules that are required for the TPT API or TPT=NO, SAS/ACCESS uses the old method of Multi-Statement insert. SAS/ACCESS can restart Multi-Statement insert from checkpoints when Multi-Statement insert uses the TPT API.

The SAS/ACCESS Multi-Statement insert facility loads both empty and existing Teradata tables. SAS/ACCESS supports only Insert operations and loading only one target table at time.

Errors are logged to Teradata tables. Error recovery can be difficult if you do not set TPT_CHECKPOINT_DATA= to enable restart from the last checkpoint. To find the error that corresponds to the code that is stored in the error table, see your Teradata documentation. You can restart a failed job for the last checkpoint by following the instructions on the SAS error log.

**Starting Multi-Statement Insert with the TPT API**

See the SAS configuration document for instructions on setting up the environment so that SAS can find the TPT API modules.

You can use one of these options to start Multi-Statement in the SAS/ACCESS interface using the TPT API:

- the TPT=YES data set option in a processing step that populates an empty Teradata table
- the TPT=YES LIBNAME option on the destination libref (the Teradata DBMS library where one or more tables are to be created and loaded)

**Multi-Statement Insert with TPT API Data Set Options**

These data set options are specific to Multi-Statement insert using the TPT API.

- TPT_PACK=
- TPT_PACKMAXIMUM=

**Legacy Load and Read Utilities**

If the TPT API is not available, SAS/ACCESS Interface to Teradata provides the following facilities. Although these are legacy facilities, they do improve performance when loading data. These correspond to native Teradata utilities.

- FastLoad
- FastExport
- MultiLoad
- Multi-Statement

For more information, see the following sections.
Using FastLoad

FastLoad Supported Features and Restrictions
SAS/ACCESS Interface to Teradata supports FastLoad, a native Teradata bulk-load utility that greatly accelerates inserting data into empty Teradata tables. For general information about using FastLoad and error recovery, see the Teradata FastLoad documentation.

The SAS/ACCESS FastLoad facility is similar to the native Teradata FastLoad Utility. They share these limitations:

- FastLoad can load only empty tables. It cannot append to a table that already contains data. If you try to use FastLoad when appending to a table that contains rows, the append step fails.
- Both the Teradata FastLoad Utility and the SAS/ACCESS FastLoad facility log data errors to tables. Error recovery can be difficult. To find the error that corresponds to the code that is stored in the error table, see the Teradata FastLoad documentation.
- FastLoad does not load duplicate rows (rows where all corresponding fields contain identical data) into a Teradata table. If your SAS data set contains duplicate rows, you can use the normal insert (load) process.
- If you create a table with special characters that require quotation marks, you must specify a name in BL_LOG= that does not require quotation marks.

Starting FastLoad
If you do not specify FastLoad, your Teradata tables are loaded normally (slowly). To start FastLoad in the SAS/ACCESS interface, you can use one of these items:

- the BULKLOAD=YES data set option in a processing step that populates an empty Teradata table
- the BULKLOAD=YES LIBNAME option on the destination libref (the Teradata DBMS library where one or more intended tables are to be created and loaded)
- the FASTLOAD= alias for either of these options

FastLoad Data Set Options
Here are the data set options that you can use with the FastLoad facility.

- BL_LOG= specifies the names of error tables that are created when you use the SAS/ACCESS FastLoad facility. By default, FastLoad errors are logged in Teradata tables named SAS_FASTLOAD_ERRS1_randnum and SAS_FASTLOAD_ERRS2_randnum, where randnum is a randomly generated number. For example, if you specify BL_LOG=my_load_errors, errors are logged in tables my_load_errors1 and my_load_errors2. If you specify BL_LOG=errtab, errors are logged in tables name errtab1 and errtab2.

  Note: SAS/ACCESS automatically deletes the error tables if no errors are logged. If errors occur, the tables are retained and SAS/ACCESS issues a warning message that includes the names of the error tables.

- DBCOMMIT=n causes a Teradata “checkpoint” after each group of n rows is transmitted. Using checkpoints slows performance but provides known synchronization points if failure occurs during the loading process. Checkpoints are not used by default if you do not explicitly set DBCOMMIT= and BULKLOAD=YES. The Teradata alias for this option is CHECKPOINT=. 
Using MultiLoad

MultiLoad Supported Features and Restrictions
SAS/ACCESS Interface to Teradata supports a bulk-load capability called MultiLoad that greatly accelerates insertion of data into Teradata tables. For general information about using MultiLoad with Teradata tables and for information about error recovery, see the Teradata MultiLoad documentation. SAS/ACCESS examples are available.

Unlike FastLoad, which only loads empty tables, MultiLoad loads both empty and existing Teradata tables. If you do not specify MultiLoad, your Teradata tables are loaded normally (inserts are sent one row at a time).

The SAS/ACCESS MultiLoad facility loads both empty and existing Teradata tables. SAS/ACCESS supports these features:

- You can load only one target table at a time.
- Only Insert and Upsert operations are supported.

Because the SAS/ACCESS MultiLoad facility is similar to the native Teradata MultiLoad utility, they share a limitation: You must drop these items onto the target tables before the load:

- unique secondary indexes
- foreign key references
- join indexes

Both the Teradata MultiLoad utility and the SAS/ACCESS MultiLoad facility log data errors to tables. Error recovery can be difficult, but the ability to restart from the last checkpoint is possible. To find the error that corresponds to the code that is stored in the error table, see the Teradata MultiLoad documentation.

MultiLoad Setup
Here are the requirements for using the MultiLoad bulk-load capability in SAS.

- The native Teradata MultiLoad utility must be present on your system. If you do not have the Teradata MultiLoad utility and you want to use it with SAS, contact Teradata to obtain the utility.
- SAS must be able to locate the Teradata MultiLoad utility on your system.
- The Teradata MultiLoad utility must be able to locate the SASMlam access module and the SasMlne exit routine. They are supplied with SAS/ACCESS Interface to Teradata.
- SAS MultiLoad requires Teradata client TTU 8.2 or later.

If not already done during post-installation configuration, see the SAS configuration documentation for your system for information about how to configure SAS to work with MultiLoad.

MultiLoad Data Set Options
Call the SAS/ACCESS MultiLoad facility by specifying MULTILOAD=YES. See the MULTILOAD= data set option for detailed information and examples on loading data and recovering from errors during the load process.
Here are the data set options that are available for use with the MultiLoad facility. For detailed information about these options, see “Overview” on page 241.

- **MBUFSIZE**=

- **ML_CHECKPOINT**=ML_CHECKPOINT=

- **ML_ERROR1**= allows the user to name the error table that MultiLoad uses for tracking errors from the acquisition phase. See the Teradata MultiLoad reference for more information about what is stored in this table. By default, the acquisition error table is named SAS_ML_ET_ranidnum where ranidnum is a random number. When restarting a failed MultiLoad job, you need to specify the same acquisition table from the earlier run so that the MultiLoad job can restart correctly. Note that the same log table, application error table, and work table must also be specified upon restarting, using ML_RESTART, ML_ERROR2, and ML_WORK data set options.

  ML_ERROR1 and ML_LOG are mutually exclusive and cannot be specified together.

- **ML_ERROR2**=

- **ML_LOG**= specifies a prefix for the temporary tables that the Teradata MultiLoad utility uses during the load process. The MultiLoad utility uses a log table, two error tables, and a work table when loading data to the target table. These tables are named by default as SAS_ML_RS_ranidnum, SAS_ML_ET_ranidnum, SAS_ML_UT_ranidnum, and SAS_ML_WT_ranidnum where ranidnum is a randomly generated number. ML_LOG is used to override the default names used. For example, if you specify ML_LOG=MY_LOAD the log table is named MY_LOAD_RS. Errors are logged in tables MY_LOAD_ET and MY_LOAD_UT. The work table is named MY_LOAD_WT.

- **ML_RESTART**= allows the user to name the log table that MultiLoad uses for tracking checkpoint information. By default, the log table is named SAS_ML_RS_ranidnum where ranidnum is a random number. When restarting a failed MultiLoad job, you need to specify the same log table from the earlier run so that the MultiLoad job can restart correctly. Note that the same error tables and work table must also be specified upon restarting the job, using ML_ERROR1, ML_ERROR2, and ML_WORK data set options. ML_RESTART and ML_LOG are mutually exclusive and cannot be specified together.

- **ML_WORK**= allows the user to name the work table that MultiLoad uses for loading the target table. See the Teradata MultiLoad reference for more information about what is stored in this table. By default, the work table is named SAS_ML_WT_ranidnum where ranidnum is a random number. When restarting a failed MultiLoad job, you need to specify the same work table from the earlier run so that the MultiLoad job can restart correctly. Note that the same log table, acquisition error table and application error table must also be specified upon restarting the job using ML_RESTART, ML_ERROR1, and ML_ERROR2 data set options. ML_WORK and ML_LOG are mutually exclusive and cannot be specified together.

- **SLEEP**= specifies the number of minutes that MultiLoad waits before it retries a logon operation when the maximum number of utilities are already running on the Teradata database. The default value is 6. SLEEP= functions very much like the SLEEP run-time option of the native Teradata MultiLoad utility.

- **TENACITY**= specifies the number of hours that MultiLoad tries to log on when the maximum number of utilities are already running on the Teradata database. The default value is 4. TENACITY= functions very much like the TENACITY run-time option of the native Teradata MultiLoad utility.

- **UPSERT**= when MultiLoad=YES, specifies that a Teradata Upsert operation should take place. The default value is NO.
• **UPSERT\_WHERE=** specifies which columns in the master table are to be used when generating a Where condition for a MultiLoad Upser. The default value is none.

• **UPSERT\_CONDITION=** specifies additional conditions to be appended to the **UPSERT\_WHERE=** option. The default value is none.

Be aware that these options are disabled while you are using the SAS/ACCESS MultiLoad facility.

• **DBCOMMIT=** LIBNAME and data set options are disabled because **DBCOMMIT=** functions very differently from **CHECKPOINT** of the native Teradata MultiLoad utility.

• A rollback does not take place to the last checkpoint on reaching **ERRLIMIT=** as the rows without errors have already been sent to Teradata.

To see whether threaded Reads are actually generated, turn on SAS tracing by setting **OPTIONS SASTRACE=",,,d"** in your program.

### Examples

This example starts the FastLoad facility.

```sas
libname fload teradata user=myusr1 password=mypwd1;
data fload.nffloat(bulkload=yes);
do x=1 to 1000000;
  output;
end;
run;
```

This next example uses FastLoad to append SAS data to an empty Teradata table and specifies the **BL\_LOG=** option to name the error tables **Append\_Err1** and **Append\_Err2**. In practice, applications typically append many rows.

```sql
/* Create the empty Teradata table. */
proc sql;
  connect to teradata as tera(user=myusr1 password=mypwd1);
  execute (create table performers
  (userid int, salary decimal(10,2), job_desc char(50)))
  by tera;
  execute (commit) by tera;
quit;

/* Create the SAS data to load. */
data local;
  input userid 5. salary 9. job_desc $50.;
datalines;
  0433 35993.00 grounds keeper
  4432 44339.92 code groomer
  3288 59000.00 manager
;
/* Append the SAS data & name the Teradata error tables. */
libname tera teradata user=myusr1 password=mypwd1;
proc append data=local base=tera.performers
  (bulkload=yes bl_log=append_err);
run;
```

This example starts the MultiLoad facility.
libname trlib teradata user=myusr1 pw=mypwd1 server=dbc;

/* Use MultiLoad to load a table with 2000 rows. */
data trlib.mlfloat(MultiLoad=yes);
   do x=1 to 2000;
      output;
   end;
run;

/* Append another 1000 rows. */
data work.testdata;
   do x=2001 to 3000;
      output;
   end;
run;

/* Append the SAS data to the Teradata table. */
proc append data=work.testdata base=trlib.mlfloat
   (MultiLoad=yes);
run;

This example loads data using TPT FastLoad.

/* Check the SAS log for this message to verify that the TPT API was used.
NOTE: Teradata connection: TPT Fastload has inserted 100 rows. */
data trlib.load(TPT=YES FASTLOAD=YES);
   do x=1 to 1000;
      output;
   end;
run;

This example restarts a MultiLoad step that recorded checkpoints and failed after
loading 2000 rows of data.
proc append data=trlib.load(TPT=YES MULTILOAD=YES
   TPT_RESTART=YES TPT_CHECKPOINT_DATA=2000)
data=work.inputdata(FIRSTOBS=2001);
run;

Autopartitioning Scheme for Teradata (Legacy)

Overview

For general information about this feature, see “Autopartitioning Techniques in
SAS/ACCESS” on page 59. For platform-specific details and special considerations, see
your Teradata documentation.

The FastExport Utility is the fastest available way to read large Teradata tables.
FastExport is software that Teradata provides that delivers data over multiple Teradata
connections or sessions. If FastExport is available, SAS threaded Reads use it. If
FastExport is not available, SAS threaded Reads generate partitioning WHERE clauses.
Using the DBSLICE= option overrides FastExport. So if you have FastExport available
and want to use it, do not use DBSLICE=. To use FastExport everywhere possible, use
DBSLICEPARM=ALL.
Note: FastExport is supported only on z/OS and UNIX. Whether automatically generated or created by using DBSLICE=, partitioning of WHERE clauses is not supported.

**FastExport and Case Sensitivity**

In certain situations Teradata returns different row results to SAS when using FastExport, compared to reading normally without FastExport. The difference arises only when all of these conditions are met:

- A WHERE clause that compares a character column with a character literal is asserted.
- The column definition is NOT CASESPECIFIC.

Unless you specify otherwise, most Teradata native utilities create NOT CASESPECIFIC character columns. On the other hand, SAS/ACCESS Interface to Teradata creates CASESPECIFIC columns. In general, this means that you do not see result differences with tables that SAS creates. However, you might see result differences with tables that Teradata utilities create, which are frequently many of your tables. To determine how Teradata creates a table, look at your column declarations with the Teradata SHOW TABLE statement.

- A character literal matches to a column value that differs only in case.

You can see differences in the rows returned if your character column has mixed-case data that is otherwise identical. For example, 'Top' and 'top' are identical except for case.

Case sensitivity is an issue when SAS generates SQL code that contains a WHERE clause with one or more character comparisons. It is also an issue when you supply the Teradata SQL yourself with the explicit SQL feature of PROC SQL. The following examples illustrate each scenario, using DBSLICEPARM=ALL to start FastExport instead of the normal SAS read:

```sas
/* SAS generates the SQL for you. */
libname trlib teradata user=myusr1 password=mypwd1 dbsliceparm=all;
proc print data=trlib.employees;
where lastname='lovell';
run;

/* Use explicit SQL with PROC SQL & supply the SQL yourself, also starting FastExport. */
proc sql;
  connect to teradata(user=myusr1 password=mypwd1 dbsliceparm=all);
  select * from connection to teradata
    (select * from sales where gender='f' and salesamt>1000);
quit;
```

For more information about case sensitivity, see your Teradata documentation.

**FastExport Password Security**

FastExport requires passwords to be in clear text. Because this poses a security risk, users must specify the full pathname so that the file path is in a protected directory:

- Windows users should specify `BL_CONTROL=“PROTECTED-DIR/myscr.ctl“`. SAS/ACCESS creates the myscr.ctl script file in the protected directory with PROTECTED-DIR as the path.
UNIX users can specify a similar pathname.

MVS users must specify a middle-level qualifier such as `BL_CONTROL="MYSCR.TEST1"` so that the system generates the USERID.MYSCR.TEST1.CTL script file.

Users can also use RACF to protect the USERID.MYSCR* profile.

**FastExport Setup**

There are three requirements for using FastExport with SAS:

- You must have the Teradata FastExport utility present on your system. If you do not have FastExport and want to use it with SAS, contact Teradata to obtain this utility.
- SAS must be able to locate the FastExport Utility on your system.
- The FastExport Utility must be able to locate the SasAxsm access module, which is supplied with your SAS/ACCESS Interface to Teradata product. SasAxsm is in the SAS directory tree, in the same location as the Teradata component.

Assuming you have the Teradata FastExport Utility, perform this setup, which varies by system:

- **Windows:** As needed, modify your Path environment variable to include both the directories containing Fexp.exe (FastExport) and SasAxsm. Place these directory specifications last in your path.

- **UNIX:** As needed, modify your library path environment variable to include the directory containing sasaxsm.sl (HP) or sasaxsm.so (Solaris and AIX). These shared objects are delivered in the `$SASROOT/sasexe` directory. You can copy these modules where you want, but make sure that the directory into which you copy them is in the appropriate shared library path environment variable. On Solaris, the library path variable is `LD_LIBRARY_PATH`. On HP-UX, it is `SHLIB_PATH`. On AIX, it is `LIBPATH`. Also, make sure that the directory containing the Teradata FastExport Utility (`fexp`), is included in the PATH environment variable. FastExport is usually installed in the `/usr/bin` directory.

- **z/OS:** No action is needed when starting FastExport under TSO. When starting FastExport with a batch JCL, the SAS source statements must be assigned to a DD name other than SYisin. This can be done by passing a parameter such as `SYisin=SASIN` in the JCL where all SAS source statements are assigned to the DD name SASIN.

Keep in mind that future releases of SAS might require an updated version of SasAxsm. Therefore, when upgrading to a new SAS version, you should update the path for SAS on Windows and the library path for SAS on UNIX.

**Using FastExport**

To use FastExport, SAS writes a specialized script to a disk that the FastExport Utility reads. SAS might also log FastExport log lines to another disk file. SAS creates and deletes these files on your behalf, so no intervention is required. Sockets deliver the data from FastExport to SAS, so you do not need to do anything except install the SasAxsm access module that enables data transfer.

On Windows, when the FastExport Utility is active, a DOS window appears minimized as an icon on your toolbar. You can maximize the DOS window, but do not close it. After a FastExport operation is complete, SAS closes the window for you.
This example shows how to create a SAS data set that is a subset of a Teradata table that uses FastExport to transfer the data:

libname trlib teradata user=myusr1 password=mypwd1;
data saslocal(keep=EMPID SALARY);
set trlib.employees(dbsliceparm=all);
run;

**FastExport and Explicit SQL**

FastExport is also supported for the explicit SQL feature of PROC SQL.

The following example shows how to create a SAS data set that is a subset of a Teradata table by using explicit SQL and FastExport to transfer the data:

```
proc sql;
connect to teradata as pro1 (user=myusr1 password=mypwd1 dbsliceparm=all);
create table saslocal as select * from connection to pro1
   (select EMPID, SALARY from employees);
quit;
```

FastExport for explicit SQL is a Teradata extension only, for optimizing Read operations, and is not covered in the threaded Read documentation.

**Exceptions to Using FastExport**

With the Teradata FastExport Utility and the SasAxsm module in place that SAS supplies, FastExport works automatically for all SAS steps where threaded Reads are enabled, except for one situation. FastExport does not handle single Access Module Processor (AMP) queries. In this case, SAS/ACCESS simply reverts to a normal single connection read. For information about FastExport and single AMP queries, see your Teradata documentation.

To determine whether FastExport worked, turn on SAS tracing in advance of the step that attempts to use FastExport. If you use FastExport, you receive this (English only) message, which is written to your SAS log:

```
teradata/tryottrm(): SELECT was processed with FastExport.
```

To turn on SAS tracing, run this statement:

```
options sastrace=',,,d' sastraceloc=saslog;
```

**Threaded Reads with Partitioning WHERE Clauses**

If FastExport is unavailable, threaded Reads use partitioning WHERE clauses. You can create your own partitioning WHERE clauses using the DBSLICE= option. Otherwise, SAS/ACCESS to Teradata attempts to generate them on your behalf. Like other SAS/ACCESS interfaces, this partitioning is based on the MOD function. To generate partitioning WHERE clauses, SAS/ACCESS to Teradata must locate a table column suitable for applying MOD. These types are eligible:

- BYTEINT
- SMALLINT
- INTEGER
- DATE
DECIMAL (integral DECIMAL columns only)

A DECIMAL column is eligible only if the column definition restricts it to integer values. In other words, the DECIMAL column must be defined with a scale of zero.

If the table that you are reading contains more than one column of the above mentioned types, SAS/ACCESS to Teradata applies some nominal intelligence to select a best choice. Top priority is given to the primary index, if it is MOD-eligible. Otherwise, preference is given to any column that is defined as NOT NULL. Because this is an unsophisticated set of selection rules, you might want to supply your own partitioning using the DBSLICE= option.

To view your table's column definitions, use the Teradata SHOW TABLE statement.

Note: Partitioning WHERE clauses, either automatically generated or created by using DBSLICE=, are not supported on z/OS. Whether automatically generated or created by using DBSLICE=, partitioning WHERE clauses is not supported on z/OS and UNIX.

FastExport versus Partitioning WHERE Clauses

Partitioning WHERE clauses are innately less efficient than FastExport. The Teradata DBMS must process separate SQL statements that vary in the WHERE clause. In contrast, FastExport is optimal because only one SQL statement is transmitted to the Teradata DBMS. However, older editions of the Teradata DBMS place severe restrictions on the system-wide number of simultaneous FastExport operations that are allowed. Even with newer versions of Teradata, your database administrator might be concerned about large numbers of FastExport operations.

Threaded Reads with partitioning WHERE clauses also place higher workload on Teradata and might not be appropriate on a widespread basis. Both technologies expedite throughput between SAS and the Teradata DBMS, but should be used judiciously. For this reason, only SAS threaded applications are eligible for threaded Read by default. To enable more threaded Reads or to turn them off entirely, use the DBSLICEPARM= option.

Even when FastExport is available, you can force SAS/ACCESS to Teradata to generate partitioning WHERE clauses on your behalf. This is accomplished with the DBI argument to the DBSLICEPARM= option (DBSLICEPARM=DBI). This feature is available primarily to enable comparisons of these techniques. In general, you should use FastExport if it is available.

The explicit SQL feature of PROC SQL supports FastExport. Partitioning WHERE clauses is not supported for explicit SQL.

Teradata Processing Tips for SAS Users

Reading from and Inserting to the Same Teradata Table

If you use SAS/ACCESS to read rows from a Teradata table and then attempt to insert these rows into the same table, you can hang (suspend) your SAS session.

Here is what happens:

- A SAS/ACCESS connection requests a standard Teradata READ lock for the Read operation.
A SAS/ACCESS connection then requests a standard Teradata WRITE lock for the Insert operation.

The WRITE lock request suspends because the read connection already holds a READ lock on the table. Consequently, your SAS session stops responding (is suspended).

Here is what happens in the next example:

SAS/ACCESS creates a read connection to Teradata to fetch the rows selected (select *) from TRA.SAMETABLE, requiring a standard Teradata READ lock; Teradata issues a READ lock.

SAS/ACCESS creates an insert connection to Teradata to insert the rows into TRA.SAMETABLE, requiring a standard Teradata WRITE lock. But the WRITE lock request suspends because the table is locked already by the READ lock.

Your SAS/ACCESS session stops responding.

libname tra teradata user=myusr1 password=mypwd1;
proc sql;
insert into tra.sametable
select * from tra.sametable;
run;

To avoid this situation, use the SAS/ACCESS locking options. For details, see “Locking in the Teradata Interface” on page 924.

Using a BY Clause to Order Query Results

SAS/ACCESS returns table results from a query in random order because Teradata returns the rows to SAS/ACCESS randomly. In contrast, traditional SAS processing returns SAS data set observations in the same order during every run of your job. If maintaining row order is important, you should add a BY clause to your SAS statements. A BY clause ensures consistent ordering of the table results from Teradata.

In this example, the Teradata ORD table has NAME and NUMBER columns. The PROC PRINT statements illustrate consistent and inconsistent ordering when it displays ORD table rows.

libname prt teradata user=myusr1 password=mypwd1;
proc print data=prt.ORD;
var name number;
run;

This statement achieves more consistent ordering because it orders PROC PRINT output by the NAME value. However, on successive runs of the statement, rows display of rows within the same number and an identical name can vary, as shown here.
The above statement always yields identical ordering because every column is specified in the BY clause. Your PROC PRINT output always looks the same.

**Using TIME and TIMESTAMP**

This example creates a Teradata table and assigns the SAS TIME8. format to the TRXTIME0 column. Teradata creates the TRXTIME0 column as the equivalent Teradata data type, TIME(0), with the value of 12:30:55.

```sas
libname mylib teradata user=myusr1 password=mypwd1;

data mylib.trxtimes;
  format trxtime0 time8.;
  trxtime0 = '12:30:55't;
run;
```

This example creates a Teradata column that specifies very precise time values. The format TIME(5) is specified for the TRXTIME5 column. When SAS reads this column, it assigns the equivalent SAS format TIME14.5.

```sas
libname mylib teradata user=myusr1 password=mypwd1;

proc sql noerrorstop;
  connect to teradata (user=myusr1 password=mypwd1);
  execute (create table trxtimes (trxtime5 time(5)
    )) by teradata;
  execute (commit) by teradata;
  execute (insert into trxtimes
    values (cast('12:12:12' as time(5))
    )) by teradata;
  execute (commit) by teradata;
quit;

/* You can print the value that is read with SAS/ACCESS. */
proc print data =mylib.trxtimes;
run;
```

SAS might not preserve more than four digits of fractional precision for Teradata TIMESTAMP.
This next example creates a Teradata table and specifies a simple timestamp column with no digits of precision. Teradata stores the value 2000-01-01 00:00:00. SAS assigns the default format DATETIME19. to the TRSTAMP0 column generating the corresponding SAS value of 01JAN2000:00:00:

```sql
proc sql noerrorstop;
  connect to teradata (user=myusr1 password=mypwd1);
  execute (create table stamps (tstamp0 timestamp(0) ) by teradata);
  execute (commit) by teradata;
  execute (insert into stamps 
    values (cast('2000–01–01 00:00:00' as timestamp(0))
    ) by teradata;
  execute (commit) by teradata;
quit;
```

This example creates a Teradata table and assigns the SAS format DATETIME23.3 to the TSTAMP3 column, generating the value 13APR1961:12:30:55.123. Teradata creates the TSTAMP3 column as the equivalent data type TIMESTAMP(3) with the value 1961-04-13 12:30:55.123.

```sas
libname mylib teradata user=myusr1 password=mypwd1;
data mylib.stamps;
  format tstamp3 datetime23.3;
  tstamp3 = '13apr1961:12:30:55.123'dt;
run;
```

This next example illustrates how the SAS engine passes the literal value for TIMESTAMP in a WHERE statement to Teradata for processing. Note that the value is passed without being rounded or truncated so that Teradata can handle the rounding or truncation during processing. This example would also work in a DATA step.

```sql
proc sql ;
  select * from trlib.flytime where col1 = '22Aug1995 12:30:00.557'dt ;
quit;
```

In SAS 8, the Teradata interface did not create TIME and TIMESTAMP data types. Instead, the interface generated FLOAT values for SAS times and dates. This example shows how to format a column that contains a FLOAT representation of a SAS datetime into a readable SAS datetime.

```sas
libname mylib teradata user=myusr1 password=mypwd1;
proc print data=mylib.stampv80;
  format stamp080 datetime25.0;
run;
```

Here, the old Teradata table STAMPV80 contains the FLOAT column, STAMP080, which stores SAS datetime values. The FORMAT statement displays the FLOAT as a SAS datetime value.

---

**Replacing PROC SORT with a BY Clause**

In general, PROC SORT steps are not useful to output a Teradata table. In traditional SAS processing, PROC SORT is used to order observations in a SAS data set. Subsequent SAS steps that use the sorted data set receive and process the observations in...
the sorted order. Teradata does not store output rows in the sorted order. Therefore, do not sort rows with PROC SORT if the destination sorted file is a Teradata table.

This example illustrates a PROC SORT statement found in typical SAS processing. You cannot use this statement in SAS/ACCESS Interface to Teradata.

```sas
libname sortprt '.';
proc sort data=sortprt.salaries;
by income;
proc print data=sortprt.salaries;
```

This next example removes the PROC SORT statement shown in the previous example. It instead uses a BY clause with a VAR clause with PROC PRINT. The BY clause returns Teradata rows ordered by the INCOME column.

```sas
libname sortprt teradata user=myusr1 password=mypwd1;
proc print data=sortprt.salaries;
var income;
by income;
```

### Reducing Workload on Teradata by Sampling

The OBS= option triggers SAS/ACCESS to add a SAMPLE clause to generated SQL. In this example, 10 rows are printed from dbc.ChildrenX:

```sas
libname tra teradata user=myusr1 pass=mypwd1 database=dbc;
proc print data=tra.ChildrenX (obs=10);
run;
```

The SQL passed to Teradata is:

```
SELECT "Child","Parent" FROM "ChildrenX"  SAMPLE 10
```

Especially against large Teradata tables, small values for OBS= reduce workload and spool space consumption on Teradata and your queries complete much sooner. See the SAMPLE clause in your Teradata documentation for more information.

### Using the Teradata Wallet Feature

#### Overview

The Teradata Wallet feature enables you to store encrypted database passwords for users without exposing those passwords within system scripts. The Teradata Wallet feature is available with TTU 14.00 and higher in UNIX or Windows environments. The Teradata Wallet feature can be used in systems that use LDAP.

Each user’s wallet contains items that consist of an item name in plain text and a corresponding encrypted value. The name serves as a hint to identify the encrypted value. For example, a wallet might contain an item with the name `uid_td2_server2` and a corresponding value that is the encrypted version of `myuserid`.

Within a script, you call the `$tdwallet` function that returns the encrypted value that the system uses in place of a text value. To assign the user ID from the previous example to a parameter that is called user, your command might contain this code:

```sas
... user="$tdwallet(uid_td2_server2)" ... 
```

In a SAS program, the values are used similarly.
For more information, including details about adding items to and deleting items from a wallet, see your Teradata documentation.

**Examples**

**Define Items in a Teradata Wallet**

Here is code to define items in a Teradata wallet on a UNIX command line. The UNIX prompt and command are on the first line. Below this line you can see the resulting prompt and message from the system:

```
machine> tdwallet add uid_td2_server2
Enter desired value for the string named "uid_td2_server2": dbtester
String named "uid_td2_server2" added.
```

The value *dbtester* is stored in an encrypted form in the Teradata wallet, and the encrypted value is passed via the $tdwallet function as "$tdwallet(uid_td2_server2)".

Here are some additional item definitions:

```
machine> tdwallet add password_td2_server2
Enter desired value for the string named "password_td2_server2": db66444333
String named "password_td2_server2" added.
```

```
machine> tdwallet add com.teradata.TD2
Enter desired value for the string named "com.teradata.TD2": $tdwallet(password_td2_$(tdpid))
String named "com.teradata.TD2" added.
```

```
machine> tdwallet add uid_ldap_server2
Enter desired value for the string named "uid_ldap_server2": joe_user
String named "uid_ldap_server2" added.
```

```
machine> tdwallet add password_ldap_server2
Enter desired value for the string named "password_ldap_server2": joe33444999
String named "password_ldap_server2" added.
```

```
machine> tdwallet add com.teradata.LDAP
Enter desired value for the string named "com.teradata.LDAP": $tdwallet(password_ldap_$(tdpid))
String named "com.teradata.LDAP" added.
```

In the item definitions for com.teradata.TD2 and com.teradata.LDAP, the argument to the $tdwallet function includes $((tdpid)). This is a reference to a predefined Teradata environment variable.

**Access Data by Using the Teradata Wallet Feature**

Use the following code to access a table that contains data about a new class. The call to the $tdwallet function with no argument automatically returns the password value for uid_td2_server2. All other calls to the $tdwallet function require arguments.

```
libname td Teradata user="$tdwallet(uid_td2_server2)" password="$tdwallet(server=server2);

proc sort data=td.class_new out=lsgout nodupkey;
    by age;
run;
```
Access Data in a Teradata Database That Has LDAP and TPT Support

The Teradata Wallet feature works with LDAP and TPT support. The call to the $tdwallet function with no argument automatically returns the password value for uid_ldap_server2. All other calls to the $tdwallet function require arguments.

Libname td.teradata user="$tdwallet(uid_ldap_server2)" password="$tdwallet(server=server2);
proc delete data=td.foo;
run;

data td.foo (multiload=yes tpt=yes);
x=55;
run;

Locking in the Teradata Interface

Overview

These LIBNAME and data set options let you control how the Teradata interface handles locking. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

Use SAS/ACCESS locking options only when Teradata standard locking is undesirable. For tips on using these options, see “Understanding SAS/ACCESS Locking Options” on page 925 and “When to Use SAS/ACCESS Locking Options” on page 926. Teradata examples are available.

READ_LOCK_TYPE= TABLE | VIEW
UPDATE_LOCK_TYPE= TABLE | VIEW
READ_MODE_WAIT= YES | NO
UPDATE_MODE_WAIT= YES | NO
READ_ISOLATION_LEVEL= ACCESS | READ | WRITE

Here are the valid values for this option.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS</td>
<td>Obtains an ACCESS lock by ignoring other users' ACCESS, READ, and WRITE locks. Permits other users to obtain a lock on the table or view. Can return inconsistent or unusual results.</td>
</tr>
<tr>
<td>READ</td>
<td>Obtains a READ lock if no other user holds a WRITE or EXCLUSIVE lock. Does not prevent other users from reading the object. Specify this isolation level whenever possible, it is usually adequate for most SAS/ACCESS processing.</td>
</tr>
</tbody>
</table>
Isolation Level | Definition
---|---
WRITE | Obtains a WRITE lock on the table or view if no other user has a READ, WRITE, or EXCLUSIVE lock on the resource. You cannot explicitly release a WRITE lock. It is released only when the table is closed. Prevents other users from acquiring any lock but ACCESS. This is unnecessarily restrictive, because it locks the entire table until the Read operation is finished.

UPDATE_ISOLATION_LEVEL = ACCESS | READ | WRITE

The valid values for this option, ACCESS, READ, and WRITE, are defined in the following table.

| Table 33.4 Update Isolation Levels for Teradata |
|---|---|
| Isolation Level | Definition |
| ACCESS | Obtains an ACCESS lock by ignoring other users' ACCESS, READ, and WRITE locks. Avoids a potential deadlock but can cause data corruption if another user is updating the same data. |
| READ | Obtains a READ lock if no other user holds a WRITE or EXCLUSIVE lock. Prevents other users from being granted a WRITE or EXCLUSIVE lock. Locks the entire table or view, allowing other users to acquire READ locks. Can lead to deadlock situations. |
| WRITE | Obtains a WRITE lock on the table or view if no other user has a READ, WRITE, or EXCLUSIVE lock on the resource. You cannot explicitly release a WRITE lock. It is released only when the table is closed. Prevents other users from acquiring any lock but ACCESS. Prevents all users, except those with ACCESS locks, from accessing the table. Prevents the possibility of a deadlock, but limits concurrent use of the table. |

These locking options cause the LIBNAME engine to transmit a locking request to the DBMS; Teradata performs all data-locking. If you correctly specify a set of SAS/ACCESS read or update locking options, SAS/ACCESS generates locking modifiers that override the Teradata standard locking.

If you specify an incomplete set of locking options, SAS/ACCESS returns an error message. If you do not use SAS/ACCESS locking options, Teradata lock defaults are in effect. For a complete description of Teradata locking, see the LOCKING statement in your Teradata SQL reference documentation.

**Understanding SAS/ACCESS Locking Options**

SAS/ACCESS locking options modify Teradata standard locking. Teradata usually locks at the row level; SAS/ACCESS lock options lock at the table or view level. The change in the scope of the lock from row to table affects concurrent access to DBMS objects. Specifically, READ and WRITE table locks increase the amount of time that other users
must wait to access the table and can decrease overall system performance. These measures help minimize these negative effects.

- Apply READ or WRITE locks only when you must apply special locking on Teradata tables.

SAS/ACCESS locking options can be appropriate for special situations, as described in “When to Use SAS/ACCESS Locking Options” on page 926. If SAS/ACCESS locking options do not meet your specialized needs, you can use additional Teradata locking features using views. See CREATE VIEW in your Teradata SQL reference documentation for details.

- Limit the span of the locks by using data set locking options instead of LIBNAME locking options whenever possible. (LIBNAME options affect all tables that you open that your libref references. Data set options apply only to the specified table.)

If you specify these read locking options, SAS/ACCESS generates and submits to Teradata locking modifiers that contain the values that you specify for the three read lock options:

- READ_ISOLATION_LEVEL= specifies the level of isolation from other table users that is required during SAS/ACCESS Read operations.
- READ_LOCK_TYPE= specifies and changes the scope of the Teradata lock during SAS/ACCESS Read operations.
- READ_MODE_WAIT= specifies during SAS/ACCESS Read operations whether Teradata should wait to acquire a lock or fail your request when the DBMS resource is locked by a different user.

If you specify these update lock options, SAS/ACCESS generates and submits to Teradata locking modifiers that contain the values that you specify for the three update lock options:

- UPDATE_ISOLATION_LEVEL= specifies the level of isolation from other table users that is required as SAS/ACCESS reads Teradata rows in preparation for updating the rows.
- UPDATE_LOCK_TYPE= specifies and changes the scope of the Teradata lock during SAS/ACCESS Update operations.
- UPDATE_MODE_WAIT= specifies during SAS/ACCESS Update operations whether Teradata should wait to acquire a lock or fail your request when the DBMS resource is locked by a different user.

When to Use SAS/ACCESS Locking Options

This section describes situations that might require SAS/ACCESS lock options instead of the standard locking that Teradata provides.

- Use SAS/ACCESS locking options to reduce the isolation level for a Read operation.

  When you lock a table using a READ option, you can lock out both yourself and other users from updating or inserting into the table. Conversely, when other users update or insert into the table, they can lock you out from reading the table. In this situation, you want to reduce the isolation level during a Read operation. To do this, you specify these read SAS/ACCESS lock options and values:

  - READ_ISOLATION_LEVEL=ACCESS
  - READ_LOCK_TYPE=TABLE
  - READ_MODE_WAIT=YES
One of these situations can result from the options and settings in this situation:

- Specify ACCESS locking, eliminating a lock out of yourself and other users. Because ACCESS can return inconsistent results to a table reader, specify ACCESS only if you are casually browsing data, not if you require precise data.

- Change the scope of the lock from row-level to the entire table.

- Request that Teradata wait if it attempts to secure your lock and finds the resource already locked.

- Use SAS/ACCESS lock options to avoid contention.

When you read or update a table, contention can occur: the DBMS is waiting for other users to release their locks on the table that you want to access. This contention suspends your SAS/ACCESS session. In this situation, to avoid contention during a Read operation, you specify these SAS/ACCESS Read lock options and values.

- \texttt{READ\_ISOLATION\_LEVEL=READ}
- \texttt{READ\_LOCK\_TYPE=TABLE}
- \texttt{READ\_MODE\_WAIT=NO}

One of these situations can result from the options and settings in this situation.

- Specify a READ lock.

- Change the scope of the lock. Because SAS/ACCESS does not support row locking when you obtain the lock requested, you lock the entire table until your Read operation finishes.

- Tell SAS/ACCESS to fail the job step if Teradata cannot immediately obtain the READ lock.

### Examples

#### Setting the Isolation Level to ACCESS for Teradata Tables

/* This generates a quick survey of unusual customer purchases. */
libname cust teradata user=myusr1 password=mypwd1
  READ\_ISOLATION\_LEVEL=ACCESS
  READ\_LOCK\_TYPE=TABLE
  READ\_MODE\_WAIT=YES
  CONNECTION=UNIQUE;

proc print data=cust.purchases(where= (bill<2));
  run;

data local;
  set cust.purchases (where= (quantity>1000));
  run;

Here is what SAS/ACCESS does in the above example.

- Connects to the Teradata DBMS and specifies the three SAS/ACCESS LIBNAME read lock options.

- Opens the PURCHASES table and obtains an ACCESS lock if a different user does not hold an EXCLUSIVE lock on the table.

- Reads and displays table rows with a value less than 2 in the BILL column.
Closes the PURCHASES table and releases the ACCESS lock.

Opens the PURCHASES table again and obtains an ACCESS lock if a different user does not hold an EXCLUSIVE lock on the table.

Reads table rows with a value greater than 1000 in the QUANTITY column.

Closes the PURCHASES table and releases the ACCESS lock.

**Setting Isolation Level to WRITE to Update a Teradata Table**

```sql
libname cust teradata user=myusr1 password=mypwd1;
proc sql;
update cust.purchases(UPDATE_ISOLATION_LEVEL=WRITE
UPDATE_MODE_WAIT=YES
UPDATE_LOCK_TYPE=TABLE)
set rebate=10 where bill>100;
quit;
```

In this example here is what SAS/ACCESS does:

- Connects to the Teradata DBMS and specifies the three SAS/ACCESS data set update lock options.
- Opens the PURCHASES table and obtains a WRITE lock if a different user does not hold a READ, WRITE, or EXCLUSIVE lock on the table.
- Updates table rows with BILL greater than 100 and sets the REBATE column to 10.
- Closes the PURCHASES table and releases the WRITE lock.

**Preventing a Hung SAS Session When Reading and Inserting to the Same Table**

```sql
libname tra teradata user=myusr1 password=mypwd1 connection=unique;
proc sql;
insert into tra.sametable
select * from tra.sametable(read_isolation_level=access
read_mode_wait=yes
read_lock_type=table);
```

Here is what SAS/ACCESS does in the above example:

- Creates a read connection to fetch the rows selected (SELECT *) from TRA.SAMETABLE and specifies an ACCESS lock (READ_ISOLATION_LEVEL=ACCESS). Teradata grants the ACCESS lock.
- Creates an insert connection to Teradata to process the Insert operation to TRA.SAMETABLE. Because the ACCESS lock that is already on the table permits access to the table, Teradata grants a WRITE lock.
- Performs the Insert operation without hanging (suspending) your SAS session.
Naming Conventions for Teradata

Teradata Conventions

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

You can use these conventions to name such Teradata objects as include tables, views, columns, indexes, and macros.

- A name must be from 1 to 32 characters long. Support for 32-character names was added in the third maintenance release for SAS 9.4.

If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name results in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or fewer. SAS does not truncate a longer table name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

Note: If a column name is longer than 32 characters, then SAS/ACCESS attempts to store the full name in the column label. The label must not contain more than 256 bytes. If the label exceeds this 256-byte limit, an error is output to the SAS log.

- A name must begin with a letter unless you enclose it in double quotation marks.

- A name can contain letters (A to Z), numbers from 0 to 9, underscore (_), dollar sign ($), and the number or pound sign (#). A name in double quotation marks can contain any characters except double quotation marks.

- A name, even when enclosed in double quotation marks, is not case sensitive. For example, CUSTOMER and Customer are the same.

- A name cannot be a Teradata reserved word.

- Because the name must be unique between objects, a view and table in the same database cannot have an identical name.

SAS Naming Conventions

Use these conventions when naming a SAS object:

- A name must be from 1 to 32 characters long.

- A name must begin with a letter (A to Z) or an underscore (_).

- A name can contain letters (A to Z), numbers from 0 to 9, and an underscore (_).

- Names are not case sensitive. For example, CUSTOMER and Customer are the same.

- A name cannot be enclosed in double quotation marks.

- A name does not need to be unique between object types.


**Naming Objects to Meet Teradata and SAS Conventions**

To easily share objects between SAS and the DBMS, create names that meet both SAS and Teradata naming conventions:

- Start with a letter.
- Include only letters, digits, and underscores.
- Use a length of 1 to 32 characters.


**Accessing Teradata Objects That Do Not Meet SAS Naming Conventions**

**Overview**

These SAS/ACCESS code examples can help you access Teradata objects (existing Teradata DBMS tables and columns) that have names that do not follow SAS naming conventions.

**Example 1: Unusual Teradata Table Name**

```sas
libname unusual teradata user=myusr1 password=mypwd1;
proc sql dquote=ansi;
    create view myview as
    select * from unusual."More names";
proc print data=myview;run;
```

**Example 2: Unusual Teradata Column Names**

SAS/ACCESS automatically converts Teradata column names that are not valid for SAS, mapping such characters to underscores. It also appends numeric suffixes to identical names to ensure that column names are unique.

```sas
create table unusual_names( Name$ char(20), Name# char(20),
"Other strange name" char(20));
```

In this example SAS/ACCESS converts the spaces found in the Teradata column name, OTHER STRANGE NAME, to Other_strange_name. After the automatic conversion, SAS programs can then reference the table as usual.

```sas
libname unusual teradata user=myusr1 password=mypwd1;
proc print data=unusual.unusual_names; run;
```

**Output 33.3  PROC PRINT Display**

<table>
<thead>
<tr>
<th>Name_</th>
<th>Name_0</th>
<th>Other_strange_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>930</td>
<td>930</td>
<td>930</td>
</tr>
</tbody>
</table>
Data Types for Teradata

Overview

Every column in a table has a name and data type. The data type tells Teradata how much physical storage to set aside for the column, as well as the form in which to store the data. This section includes information about Teradata data types, null values, and data conversions.

Supported Teradata Data Types

Here are the data types that SAS/ACCESS Interface to Teradata supports:

- Binary string data:
  
  BYTE(n)
  VARBYTE(n)

- Character string data:

  CHAR (n)
  VARCHAR (n)
  LONG VARCHAR

- Date and time data:

  DATE
  TIME(n)
  TIMESTAMP (n)

  Date type columns might contain Teradata values that are out of range for SAS, which handles dates from A.D. 1582 through A.D. 20,000. If SAS/ACCESS encounters an unsupported date (for example, a date earlier than A.D. 1582), it returns an error message and displays the date as a missing value.

  Note: When processing WHERE statements (using PROC SQL or the DATA step) that contain literal values for TIME or TIMESTAMP, the SAS engine passes the values to Teradata exactly as they were entered, without being rounded or truncated. This is done so that Teradata can handle the rounding or truncation during processing.

- Numeric data:

  BYTEINT
  INTEGER
  DECIMAL(n,m)
  SMALLINT
  FLOAT | REAL | DOUBLE PRECISION

  Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character
data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

**Teradata Null Values**

Teradata has a special value that is called NULL. A Teradata NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Teradata NULL value, it interprets it as a SAS missing value.

By default, Teradata columns accept NULL values. However, you can define columns so that they do not contain NULL values. For example, when you create a SALES table, define the CUSTOMER column as NOT NULL. This tells Teradata not to add a row to the table unless the CUSTOMER column for the row has a value. When creating a Teradata table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how SAS missing character values are handled, use the NULLCHAR= and NULLCHARVAL= data set options.

**LIBNAME Statement Data Conversions**

This table shows the default formats that SAS/ACCESS Interface to Teradata assigns to SAS variables when using the LIBNAME statement to read from a Teradata table. SAS/ACCESS does not use Teradata table column attributes when it assigns defaults.

<table>
<thead>
<tr>
<th>Teradata Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR((n))</td>
<td>$n (n&lt;=32,767)</td>
</tr>
<tr>
<td>CHAR((n))</td>
<td>$32767.(n&gt;32,767) *</td>
</tr>
<tr>
<td>VARCHAR((n))</td>
<td>$n (n&lt;=32,767)</td>
</tr>
<tr>
<td>VARCHAR((n))</td>
<td>$32767.(n&gt;32,767) *</td>
</tr>
<tr>
<td>LONG VARCHAR((n))</td>
<td>$32767. *</td>
</tr>
<tr>
<td>BYTE((n))</td>
<td>$HEXn. (n&lt;=32,767)</td>
</tr>
<tr>
<td>BYTE((n)) *</td>
<td>$HEX32767.(n&gt;32,767)</td>
</tr>
<tr>
<td>VARBYTE((n))</td>
<td>$HEXn. (n&lt;=32,767)</td>
</tr>
<tr>
<td>VARBYTE((n))</td>
<td>$HEX32767.(n&gt;32,767)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>11.0</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>6.0</td>
</tr>
</tbody>
</table>
When reading Teradata data into SAS, DBMS columns that exceed 32,767 bytes are truncated. The maximum size for a SAS character column is 32,767 bytes.

** If the DECIMAL number is extremely large, SAS can lose precision. For details, see “Supported Teradata Data Types” on page 931.

*** To learn how SAS/ACCESS handles dates that are outside the valid SAS date range, see “Supported Teradata Data Types” on page 931.

† TIME and TIMESTAMP are supported for Teradata Version 2, Release 3, and later. The TIME with TIMEZONE, TIMESTAMP with TIMEZONE, and INTERVAL types are presented as SAS character strings and are therefore harder to use.

When you create Teradata tables, the default Teradata columns that SAS/ACCESS creates are based on the type and format of the SAS column. The following table shows the default Teradata data types that SAS/ACCESS assigns to the SAS formats during output processing when you use the LIBNAME statement.

### Table 33.6 Default Output Teradata Data Types

<table>
<thead>
<tr>
<th>SAS Data Type</th>
<th>SAS Format</th>
<th>Teradata Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>$w:</td>
<td>CHAR[w]</td>
</tr>
<tr>
<td></td>
<td>$CHAR[w]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$VARYING[w]</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>SHEXw:</td>
<td>BYTE[w]</td>
</tr>
<tr>
<td>Numeric</td>
<td>A date format</td>
<td>DATE</td>
</tr>
<tr>
<td>Numeric</td>
<td>TIMEw:d</td>
<td>TIME(d)</td>
</tr>
</tbody>
</table>
### SAS Data Type

<table>
<thead>
<tr>
<th>SAS Data Type</th>
<th>SAS Format</th>
<th>Teradata Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>DATETIME(w.d)</td>
<td>TIMESTAMP(d)</td>
</tr>
<tr>
<td>Numeric</td>
<td>(w: (w\leq2))</td>
<td>BYTEINT</td>
</tr>
<tr>
<td>Numeric</td>
<td>(w: (3\leq w\leq4))</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Numeric</td>
<td>(w: (5\leq w\leq9))</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Numeric</td>
<td>(w: (6\leq w\leq10))</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>Numeric</td>
<td>(w: (w\geq19))</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Numeric</td>
<td>(w.d)</td>
<td>DECIMAL(w-1,d)</td>
</tr>
<tr>
<td>Numeric</td>
<td>all other numeric formats</td>
<td>FLOAT</td>
</tr>
</tbody>
</table>

To override any default output type, use the DBTYPE= data set option.

#### Data Returned as SAS Binary Data with Default Format $HEX
- BYTE
- VARBYTE
- LONGVARBYTE
- GRAPHIC
- VARGRAPHIC
- LONG VARGRAPHIC

### Temporal Data Types for Teradata

#### Overview

Teradata provides built-in support for temporal data. Temporal data makes it easy to build and maintain applications where information changes over time. For example, consider a merchandise application that must store the price of an item along with the dates during which that price is valid. As shown below, traditional table design uses two date columns to store the beginning and end dates during which the price is valid.

```sql
CREATE TABLE price
(
    Item_UPC    BIGINT,
    Price       FLOAT,
    Begin_date  DATE,
    End_Date    DATE
)
```
When the price of the item changes, the end date of the current price must be updated, and a new row must be inserted with the new price.

<table>
<thead>
<tr>
<th>Item_UPC</th>
<th>Price</th>
<th>Begin_Date</th>
<th>End_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789012</td>
<td>$5</td>
<td>2010-01-05</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

Teradata temporal support can simplify this process. You can use the PERIOD(DATE) data type to represent the time period during which the price is valid. Here is how you can create the table instead.

```
CREATE TABLE price
(
    Item_UPC       BIGINT,
    Price          FLOAT,
    Price_validity PERIOD(DATE) NOT NULL AS VALIDTIME
)
```

<table>
<thead>
<tr>
<th>Item_UPC</th>
<th>Price</th>
<th>Price_validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789012</td>
<td>$5</td>
<td>(2010-01-05, 9999-12-31)</td>
</tr>
</tbody>
</table>

For additional examples and information about features, see the Teradata temporal table support documentation.

### Supported Temporal Data Types

SAS/ACCESS Interface to Teradata supports these temporal data types.

- PERIOD(TIME)
- PERIOD(DATE)
- PERIOD(TIMESTAMP)
- PERIOD(DATE) VALIDTIME/TRANSACTIONTIME
- PERIOD(TIMESTAMP) VALIDTIME/TRANSACTIONTIME

### Specifying Transaction Time and Valid Time

For true temporal support, you must specify the transaction-time and valid-time attributes on the PERIOD data type.

The transaction-time attribute on a PERIOD column makes the table a transaction-time table. Teradata automatically maintains tables with transaction-time columns. It tracks when a row is first made known to the table. When a row is inserted, it is considered to be an open row because it is currently in effect until the end of time. If the row is deleted, Teradata marks it as a closed row that is no longer in effect. However, the table can be queried to obtain rows that were open at a particular point in time even though the row is not currently valid. Similarly, when a row is modified, the current row is closed and a new row is opened and made effective.
A user cannot set or modify a transaction-time column.

The valid-time attribute indicates the time period during which the information is in effect. If valid time is specified with the PERIOD data type, Teradata maintains how the time period is in effect if the row is updated or deleted. As in the example about prices, when a row is inserted with a new price, Teradata maintains the end date of the original row. The row with the old price is updated with an end date and the new row is inserted.

A row in a valid-time transaction table can be a history row, a current row, or a future row. The history row is no longer valid with respect to current time. Its end-time period is before the current time. A current row has a time period that straddles the current time.

Creating a Table from SAS with the PERIOD Data Type

To create a Teradata table with temporal data types from SAS, use the DBTYPE= data set option. SAS does not have an equivalent data type for PERIOD. The value is represented in SAS as a character string. In this example, when the character string that represents the period ID is inserted into Teradata, it is implicitly converted to a PERIOD data type.

```sas
data x.mytest(DBTYPE=(validity='PERIOD(DATE) VALIDTIME'));
  i=1;
  validity='(1973-02-03, 9999-12-31)';
  output;
run;
```

Reading in a PERIOD Data Type

A Teradata PERIOD data-type column can be read into SAS like any other column. It is represented in SAS as a character string, such as '(1973-02-03, 9999-12-31)'.

Temporal Qualifiers

Temporal tables contain rows that can be current, history, or future in the valid-time dimension. In the transaction-time dimension, rows can be open or closed. Temporal qualifiers specify what data is needed. The TEMPORAL_QUALIFIER=LIBNAME and data set options let you qualify queries in the valid-time or transaction-time dimension. For example, to fetch rows that are valid as of '2009-01-01' in a table, you must specify TEMPORAL_QUALIFIER='VALIDTIME AS OF DATE '2009-01-01' ' as a LIBNAME or data set option when you query temporal tables.

The option that you specify for TEMPORAL_QUALIFIER= is free-form text. Here are some examples.

```
TEMPORAL_QUALIFIER='CURRENT VALIDTIME'
TEMPORAL_QUALIFIER='CURRENT TRANSACTIONTIME'
TEMPORAL_QUALIFIER='NONSEQUENCED VALIDTIME'
TEMPORAL_QUALIFIER='SEQUENCED VALIDTIME'
TEMPORAL_QUALIFIER='NONSEQUENCED VALIDTIME PERIOD '2007-01-01, 2008-03-01)'
TEMPORAL_QUALIFIER='SEQUENCED VALIDTIME PERIOD '(2007-01-01, 2008-03-01)'
TEMPORAL_QUALIFIER='CURRENT TRANSACTIONTIME'
TEMPORAL_QUALIFIER='TRANSACTIONTIME AS OF TIMESTAMP '2009-01-01 01:02:03.123456'
```

If you specify the temporal qualifier on the LIBNAME, it applies to the entire session because it is implemented by issuing session commands at connect time. For example, if you specify TEMPORAL_QUALIFIER='ASOF PERIOD '1999-01-01,
on the LIBNAME, here is the Teradata SET SESSION command that is issued at connect time. The SQL is submitted as usual.

```
.SET SESSION ASOF PERIOD '(1999-01-01, 2099-01-05)'
```

If you submit the above command, the temporal qualifier is added as a prefix, as shown below.

```
ASOF PERIOD '(1999-01-01, 2099-01-05)'
SELECT * from TEMPORAL_TABLE;
```

Sample code
-------------

```
/* PERIOD data types require the Teradata V13 server. */
libname x teradata user=myusr1 pw=mypwd1 server=mysrv1;

/* Create a table with the PERIOD(DATE) data type. 
   Note:  This is not a temporal table. */
data x.mytest(DBTYPE=(validity='PERIOD(DATE)'));
i=1; validity='(1973-02-03, 9999-12-31)'; output;
run;

/* Read from a table with a PERIOD data type? */
proc print data=x.mytest;
run;

/* Use Fastload to load a table with a PERIOD data type. */
proc datasets library=x;
   delete mytest;run;

data x.mytest(DBTYPE=(validity='PERIOD(TIMESTAMP)') FASTLOAD=YES TPT=NO);
i=1; validity='(1970-01-05 01:02:03.123, 1970-01-05 05:06:07.456)'; output;
run;

/* Temporal support starts in Teradata V13.10. */
libname x teradata user=myusr1 pw=mypwd1 server=mysrv1;

/* Create a table with the PERIOD(DATE) data type. */
data x.mytest(DBTYPE=(validity='PERIOD(DATE) VALIDTIME'));
i=1; validity='(1973-02-03, 1999-12-31)'; output;
i=2; validity='(2000-01-01, 2011-01-01)'; output;
i=3; validity='(2011-01-02, 9999-12-31)'; output;
run;

/* Can we read a PERIOD data type? 
   You must select the row with i=2. */
proc print data=x.mytest(TEMPORAL_QUALIFIER='CURRENT VALIDTIME');
run;

/* Consider data as of 1995-01-01. */
libname x teradata user=myusr1 pw=mypwd1 server=mysrv1
   TEMPORAL_QUALIFIER='VALIDTIME AS OF DATE '1995-01-01' '
/* Row with i=1 is returned. */
proc print data=x.mytest(DBSLICEPARM=ALL);
run;
```
Chapter 34
SAS/ACCESS Interface to Vertica

Introduction to SAS/ACCESS Interface to Vertica ........................................ 939
LIBNAME Statement Specifics for Vertica .................................................... 940
  Overview ........................................................................................................ 940
  Arguments ..................................................................................................... 940
  Vertica LIBNAME Statement Examples ....................................................... 943
Data Set Options for Vertica ................................................................. 943
SQL Pass-Through Facility Specifics for Vertica ........................................... 945
  Key Information .......................................................................................... 945
  CONNECT Statement Example .................................................................. 946
Understanding Vertica Update and Delete Rules ............................................. 947
Passing SAS Functions to Vertica .............................................................. 947
Passing Joins to Vertica .................................................................................. 948
Locking in the Vertica Interface ................................................................. 949
Naming Conventions for Vertica ................................................................. 950
Data Types for Vertica .................................................................................... 951
  Overview ........................................................................................................ 951
  Supported Vertica Data Types .................................................................. 951
  Vertica Null Values ...................................................................................... 952

Introduction to SAS/ACCESS Interface to Vertica

For available SAS/ACCESS features, see Vertica supported features on page 90. For more information about Vertica, see your Vertica documentation.

Note: The SAS/ACCESS Interface to Vertica was implemented for SAS 9.4.
LIBNAME Statement Specifics for Vertica

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Vertica supports. For general information about this feature, see LIBNAME Statement for Relational Databases on page 96.

Here is the LIBNAME statement syntax for accessing Vertica.

LIBNAME libref vertica <connection-options> <LIBNAME-options>;

Arguments

libref

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

vertica

specifies the SAS/ACCESS engine name for the Vertica interface.

connection-options

provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. When you use the LIBNAME statement, you can connect to the Vertica server in one of two ways. Specify only one of these methods for each connection because they are mutually exclusive.

- SERVER=, DATABASE=, PORT=, USER=, PASSWORD=
- DSN=, USER=, PASSWORD=

Here is how these options are defined.

SERVER=<>Vertica-server-name<> specifies the server name or IP address of the Vertica server to which you want to connect. This server accesses the database that contains the tables and views that you want to access. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

DATABASE=<>Vertica-database-name<> specifies the name of the database on the Vertica server that contains the tables and views that you want to access. If the database name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: DB=

PORT=port specifies the port number that is used to connect to the specified Vertica server. If you do not specify a port, the default is 5433.

USER=<>Vertica-user-name<> specifies the Vertica user name (also called the user ID) that you use to connect to your database. If the user name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: UID
PASSWORD=<'Vertica-password'>

specifies the password that is associated with your Vertica user name. If the password contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: PWD=

DSN=<'Vertica-data-source'>

specifies the configured Vertica ODBC data source to which you want to connect. Use this option if you have existing Vertica ODBC data sources that are configured on your client. This method requires additional setup. This extra setup is done through the ODBC Administrator control panel on Windows platforms, through the odbc.ini file, or a similarly named configuration file on UNIX platforms. Therefore, it is recommended that you use this connection method only if you have existing, functioning data sources that have been defined.

**LIBNAME-options**

define how SAS processes DBMS objects. Some LIBNAME options can enhance performance; others determine locking or naming behavior. The following table describes the LIBNAME options for SAS/ACCESS Interface to Vertica, with the applicable default values. For details, see LIBNAME Options for Relational Databases on page 101.

**Table 34.1 SAS/ACCESS LIBNAME Options for Vertica**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>AUTOCOMMIT=</td>
<td>NO</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>UNIQUE</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCLIENT_MAX_BYTES=</td>
<td>matches the maximum number of bytes per single character of the SAS session encoding</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 when inserting rows, 0 when updating rows</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>YES</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td><code>DLIBTERM=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBMAX_TEXT=</code></td>
<td>1024</td>
</tr>
<tr>
<td><code>DBMSTEMP=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBNULLKEYS=</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>DBPROMPT=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBSASLABEL=</code></td>
<td>COMPAT</td>
</tr>
<tr>
<td><code>DBSERVER_MAX_BYTES=</code></td>
<td>usually 1</td>
</tr>
<tr>
<td><code>DBSLICEPARAM=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DEFER=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DELETE_MULT_ROWS=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DIRECT_EXE=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DIRECT_SQL=</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>IGNORE_READ_ONLY_COLUMNS=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>INSERT_SQL=</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>INSERTBUFF=</code></td>
<td>based on row length</td>
</tr>
<tr>
<td><code>KEYSET_SIZE=</code></td>
<td>0</td>
</tr>
<tr>
<td><code>MULTI_DATASRC_OPT=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>PRESERVE_COL_NAMES=</code></td>
<td>see “Data Types for Vertica”</td>
</tr>
<tr>
<td><code>PRESERVE_TAB_NAMES=</code></td>
<td>see “Data Types for Vertica”</td>
</tr>
<tr>
<td><code>QUALIFIER=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>QUERY_TIMEOUT=</code></td>
<td>0</td>
</tr>
<tr>
<td><code>QUOTE_CHAR=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>READ_ISOLATION_LEVEL=</code></td>
<td>RC (see “Locking in the Vertica Interface”)</td>
</tr>
<tr>
<td><code>READ_LOCK_TYPE=</code></td>
<td>ROW</td>
</tr>
<tr>
<td><code>READBUFF=</code></td>
<td>1</td>
</tr>
</tbody>
</table>
### Vertica LIBNAME Statement Examples

No DSN is specified in this example. This example uses the recommended default settings for the connection options to make the connection.

```sql
libname mydblib vertica server="mysrv1" port=5433
    user=myusr1 password=mypwd1 database=mydb1;
```

A DSN is specified in this next example.

```sql
libname mydblib vertica dsn=mydsn1
    user=myusr1 password=mypwd1;
```

### Data Set Options for Vertica

All SAS/ACCESS data set options in this table are supported for Vertica. Default values are provided where applicable. For details, see Data Set Options for Relational Databases on page 241.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REREAD_EXPOSURE=</td>
<td>NO</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>none</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS=</td>
<td>none</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>RC (see “Locking in the Vertica Interface”)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>ROW</td>
</tr>
<tr>
<td>UPDATE_MULT_ROWS=</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE_SQL=</td>
<td>YES</td>
</tr>
<tr>
<td>UTILCONN_TRANSIENT=</td>
<td>NO</td>
</tr>
</tbody>
</table>
### Table 34.2 SAS/ACCESS Data Set Options for Vertica

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DBCOMMIT=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBCONDITION=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBCREATE_TABLE_OPTS=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBFORCE=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBGEN_NAME=</code></td>
<td>DBMS</td>
</tr>
<tr>
<td><code>DBINDEX=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBKEY=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBLABEL=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>DBMASTER=</code></td>
<td>none</td>
</tr>
<tr>
<td><code>DBMAX_TEXT=</code></td>
<td>1024</td>
</tr>
<tr>
<td><code> DBNULL=</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>DBNULLKEYS=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBPROMPT=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>DBSASLABEL=</code></td>
<td>COMPAT</td>
</tr>
<tr>
<td><code>DBSASTYPE=</code></td>
<td>see “Data Types for Vertica” on page 951</td>
</tr>
<tr>
<td><code>DBSLICE=</code></td>
<td>none (see “Data Types for Vertica” on page 951)</td>
</tr>
<tr>
<td><code>DBSLICEPARM=</code></td>
<td>NONE</td>
</tr>
<tr>
<td><code>DBTYPE=</code></td>
<td>none (see “Data Types for Vertica” on page 951)</td>
</tr>
<tr>
<td><code>ERRLIMIT=</code></td>
<td>1</td>
</tr>
<tr>
<td><code>IGNORE_READ_ONLY_COLUMNS=</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>INSERT_SQL=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>INSERTBUFF=</code></td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td><code>NULLCHAR=</code></td>
<td>SAS</td>
</tr>
<tr>
<td><code>NULLCHARVAL=</code></td>
<td>a blank character</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>POST_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>POST_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_STMT_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUALIFIER=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>QUERY_TIMEOUT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READ_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting (see “Locking in the Vertica Interface” on page 949)</td>
</tr>
<tr>
<td>READ_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>READBUFF=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_ISOLATION_LEVEL=</td>
<td>LIBNAME option setting (see “Locking in the Vertica Interface” on page 949)</td>
</tr>
<tr>
<td>UPDATE_LOCK_TYPE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>UPDATE_SQL=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>

**SQL Pass-Through Facility Specifics for Vertica**

**Key Information**

For general information about this feature, see “SQL Pass-Through Facility” on page 471.

Here are the SQL pass-through facility specifics for the Vertica interface.

- The `dbms-name` is `VERTICA`.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to Vertica. If you use multiple simultaneous connections, you must use the `alias` argument to identify the different connections. If you do not specify an alias, the default `VERTICA` alias is used.
• The database-connection-arguments for the CONNECT statement are identical to its LIBNAME connection options.

• These LIBNAME options are also available with the CONNECT statement.
  - AUTHDOMAIN=
  - AUTOCOMMIT=
  - DBPROMPT=
  - DEFER=
  - DELETE_MULT_ROWS=
  - IGNORE_READ_ONLY_COLUMNS=
  - INSERTBUFF=
  - QUERY_TIMEOUT=
  - READBUFF=
  - READ_ISOLATION_LEVEL=
  - STRINGDATES=
  - TRACE=
  - TRACEFILE=
  - UPDATE_ISOLATION_LEVEL=
  - UPDATE_MULTI_ROWS=
  - UTILCONN_TRANSIENT=

**CONNECT Statement Example**

This example uses the DBCON alias to connect to the mydb1 Vertica database, execute several queries, and then disconnect. The connection alias is optional.

```
proc sql;
  connect to vertica as dbcon
    (server=mysrv1 port=5433 user=myusr1 password=mypwd1 db=mydb1);
  
  execute (drop table scnpastbl) by dbcon;
  execute (create table scnpastbl (id int,
      name char(13), tel char(11), constraint_vpk
      primary key(id, tel))) by dbcon;
  execute (insert into scnpastbl values
      (1, '111', '1-1-1')) by dbcon;
  
  select * from connection to dbcon
    (select * from scnpastbl);

  disconnect from dbcon;
quit;
```
Understanding Vertica Update and Delete Rules

To avoid data integrity problems when updating or deleting data, you must define a primary key on your table.

This example uses DBTYPE= to create the primary key.

libname invty vertica server=mysrv1 port=5433 user=myusr1 database=mydb1;

proc sql;
drop table invty.STOCK23;
quit;

data invty.STOCK23(DBTYPE=(RECDATE="date not null,primary key(RECDATE)");
input PARTNO $ DESCX $ INSTOCK @17
   RECDATE date7. @25 PRICE;
format RECDATE date7. .
datalines;
   K89R  seal     34  27jul95  245.00
   M447  sander   98  20jun95   45.88
   LK43  filter  121  19may96   10.99
   MN21  brace  43  10aug96   27.87
   BC85  clamp  80  16aug96    9.55
   KJ66  cutter  6  20mar96   24.50
   UYN7  rod  211  18jun96   19.77
   JD03  switch 383  09jan97   13.99
   BV1I  timer  26  03jan97   34.50
;

These next examples show how you can update the table now that STOCK23 has a primary key.

proc sql;
update invty.STOCK23 set price=price*1.1 where INSTOCK > 50;
quit;

proc sql;
delete from invty.STOCK23 where  INSTOCK > 150;
quit;

Note: Vertica does not enforce uniqueness of primary keys when they are loaded into a table. Instead, consider using sequences or auto-incrementing of columns for primary-key columns. It guarantees uniqueness and avoids the problem of constraint enforcement and the associated overhead. For more information about sequencing, see your Vertica documentation.

Passing SAS Functions to Vertica

SAS/ACCESS Interface to Vertica passes the following SAS functions to Vertica for processing. Where the Vertica function name differs from the SAS function name, the
Vertica name appears in parentheses. For more information, see “Passing Functions to the DBMS Using PROC SQL” on page 44.

ABS
ARCOS (ACOS)
ARSIN (ASIN)
ATAN
ATAN2
AVG
BYTE
CEIL
COALESCE
COS
EXP
FLOOR
HOUR
INDEX (STRPOS)
LENGTH
LENGTHC (CHARACTER_LENGTH)
LENGTHN (LENGTH)
LOG (LN)
LOG10 (LOG)
LOWCASE (LOWER)
MAX
MIN
MINUTE
MONTH
QTR (QUARTER)
REPEAT
SECOND
SIGN
SIN
SQRT
STD
STDDEV_SAMP
STRIP (BTRIM)
SUBSTR
TAN
TRANWRD (REPLACE)
TRIMN (RTRIM)
UPCASE (UPPER)
VAR (VAR_SAMP)
YEAR

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding database functions that are passed down to the database. Only when SQL_FUNCTIONS=ALL can the SAS/ACCESS engine also pass these SAS SQL functions to Vertica. Due to incompatibility in date and time functions between Vertica and SAS, Vertica might not process them correctly. Check your results to determine whether these functions are working as expected.

COMPRESS (TRANSLATE)
DATE (CURRENT_DATE)
DATEPART (DATE)
DATETIME (GETDATE)
DAY (DAYOFMONTH)
DTEXT (DAYOFWEEK)
DTEXTWEEKDAY (DAYOFWEEK)
TIME (CLOCK_TIMESTAMP)
TIMEPART (CAST)
TODAY (CURRENT_DATE)
TRANSLATE
WEEKDAY (DAYOFWEEK)

Passing Joins to Vertica

For a multiple libref join to pass to Vertica, all of these components of the LIBNAME statements must match exactly:

• user ID (USER=)
• password (PASSWORD=)
• server (SERVER=)
Locking in the Vertica Interface

These LIBNAME and data set options let you control how the Vertica interface handles locking. For general information about an option, see “LIBNAME Options for Relational Databases” on page 101.

READ_LOCK_TYPE=ROW
UPDATE_LOCK_TYPE=ROW
READ_ISOLATION_LEVEL=S | RR | RC | RU | V

The ODBC driver manager supports the S, RR, RC, RU, and V isolation levels that are defined in this table.

Table 34.3  Isolation Levels for Vertica

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (serializable)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads.</td>
</tr>
<tr>
<td>RR (repeatable read)</td>
<td>Does not allow dirty Reads or nonrepeatable Reads; does allow phantom Reads.</td>
</tr>
<tr>
<td>RC (read committed)</td>
<td></td>
</tr>
<tr>
<td>RU (read uncommitted)</td>
<td>Allows dirty Reads, nonrepeatable Reads, and phantom Reads,</td>
</tr>
<tr>
<td>V (versioning)</td>
<td>Does not allow dirty Reads, nonrepeatable Reads, or phantom Reads. These transactions are serializable but higher concurrency is possible than with the serializable isolation level. Typically, a nonlocking protocol is used.</td>
</tr>
</tbody>
</table>

Here are how the terms in the table are defined.

Dirty reads

A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it can see changes that are made by those concurrent transactions even before they commit.

For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.
Phantom reads

When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist, a phantom.

`UPDATE_ISOLATION_LEVEL=S | RR | RC | V`

The ODBC driver manager supports the S, RR, RC, and V isolation levels defined in the preceding table.

---

### Naming Conventions for Vertica

For general information, see Chapter 2, “SAS Names and Support for DBMS Names,” on page 13.

Most SAS names can be up to 32 characters long. The Vertica interface supports table names and column names that contain up to 32 characters. If DBMS column names are longer than 32 characters, they are truncated to 32 characters. If truncating a column name would result in identical names, SAS generates a unique name by replacing the last character with a number. DBMS table names must be 32 characters or less. SAS does not truncate a longer name. If you already have a table name that is greater than 32 characters, it is recommended that you create a table view.

The `PRESERVE_TAB_NAMES=` and `PRESERVE_COL_NAMES=` options determine how this interface handles case sensitivity, spaces, and special characters. (For information about these options, see LIBNAME Statement for Relational Databases on page 96.) Vertica is not case sensitive, and all names default to lowercase.

Vertica objects include tables, views, and columns. Follow these naming conventions:

- A name must be from 1 to 128 characters long.
- A name must begin with a letter (A through Z), diacritic marks, or non-Latin characters (200-377 octal)
- A name cannot begin with an underscore (\_). Leading underscores are reserved for system objects.
- Names are not case sensitive. For example, `CUSTOMER` and `Customer` are the same, but object names are converted to lowercase when they are stored in the Vertica database. However, if you enclose a name in quotation marks, it is case sensitive.
- A name cannot be a Vertica reserved word, such as WHERE or VIEW.
- A name cannot be the same as another Vertica object that has the same type.
Data Types for Vertica

Overview

Every column in a table has a name and a data type. The data type tells Vertica how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Vertica data types, null and default values, and data conversions.

For more information about Vertica data types and to determine which data types are available for your version of Vertica, see your Vertica documentation.

SAS/ACCESS Interface to Vertica does not directly support any data types that are not listed below. Any columns using these types are read into SAS as character strings.

Supported Vertica Data Types

Here are the data types that the Vertica engine supports.

- Binary string data: BINARY
- String data:
  - CHAR
  - VARCHAR
- Numeric data:
  - BOOLEAN
  - INT8
  - DOUBLE PRECISION
  - SMALINT
  - FLOAT
  - TINYINT
  - FLOAT(n)
  - DECIMAL
  - FLOAT8
  - NUMERIC
  - REAL
  - NUMBER
  - INTEGER | INT
  - MONEY
  - BIGINT

Note: When performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that SAS reads are rounded to meet this condition. When you use a large numeric value in a WHERE clause, this rounding can cause unexpected results, such as not selecting desired rows. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data. For more information, see “Your Options When Choosing the Degree of Precision That You Need” on page 10.

- Date, time, and timestamp data:
  - DATE
  - TIME WITH TIMEZONE
  - DATETIME
  - TIMESTAMP
  - SMALLDATETIME
  - TIMESTAMP WITH TIMEZONE
  - TIME
  - INTERVAL
Vertica Null Values

Many relational database management systems have a special value called NULL. A DBMS NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DBMS NULL value, it interprets it as a SAS missing value.

In most relational databases, columns can be defined as NOT NULL so that they require data (they cannot contain NULL values). When a column is defined as NOT NULL, the DBMS does not add a row to the table unless the row has a value for that column. When creating a DBMS table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

For more information about how SAS handles NULL values, see “Potential Result Set Differences When Processing Null Data” on page 33.

To control how the DBMS handles SAS missing character values, use the NULLCHAR= and NULLCHARVAL= data set options.
Part 4

Appendixes

Appendix 1
ACCESS Procedure .................................................. 955

Appendix 2
CV2VIEW Procedure .................................................. 975

Appendix 3
DBLOAD Procedure .................................................. 985
Overview: ACCESS Procedure

Accessing DBMS Data

SAS still supports this legacy procedure. However, to access your relational DBMS data more directly, it is recommended that you use the SAS/ACCESS LIBNAME statement.
on page 96 or the SQL pass-through facility on page 472. To determine whether this feature is available in your environment for your interface, see SAS/ACCESS Features by Host on page 77.

With the DBLOAD procedure and an interface view engine, the ACCESS procedure creates an interface between SAS and data in vendor databases. You can use this procedure to create and update descriptors.

About ACCESS Procedure Statements

This procedure has several types of statements:

- **database connection statements**: for connecting to your DBMS (see DBMS-specific information in this document for your SAS/ACCESS interface)
- **creating and updating statements**: `CREATE` and `UPDATE`
- **table and editing statements**: `ASSIGN`, `DROP`, `FORMAT`, `LIST`, `QUIT`, `RENAME`, `RESET`, `SELECT`, `SUBSET`, `TABLE=` and `UNIQUE`

This table summarizes PROC ACCESS options and statements that are required to accomplish common tasks.

**Table A1.1 Statement Sequence for Accomplishing Tasks with the ACCESS Procedure**

<table>
<thead>
<tr>
<th>Task</th>
<th>Statements and Options to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an access descriptor</td>
<td><code>PROC ACCESS statement-options;</code> &lt;br&gt; <code>CREATE libref.member-name.ACCESS;</code> &lt;br&gt; <code>database-connection-statements;</code> &lt;br&gt; <code>editing-statements;</code> &lt;br&gt; <code>RUN;</code></td>
</tr>
<tr>
<td>Create an access descriptor and a view descriptor</td>
<td><code>PROC ACCESS statement-options;</code> &lt;br&gt; <code>CREATE libref.member-name.ACCESS;</code> &lt;br&gt; <code>database-connection-statements;</code> &lt;br&gt; <code>editing-statements;</code> &lt;br&gt; <code>CREATE libref.member-name.VIEW;</code> &lt;br&gt; <code>SELECT column-list;</code> &lt;br&gt; <code>editing-statements;</code> &lt;br&gt; <code>RUN;</code></td>
</tr>
<tr>
<td>Create a view descriptor from an existing access descriptor</td>
<td><code>PROC ACCESS statement-options, including ACCDESC=libref.access-descriptor;</code> &lt;br&gt; <code>CREATE libref.member-name.VIEW;</code> &lt;br&gt; <code>SELECT column-list;</code> &lt;br&gt; <code>editing-statements;</code> &lt;br&gt; <code>RUN;</code></td>
</tr>
<tr>
<td>Update an access descriptor</td>
<td><code>PROC ACCESS statement-options;</code> &lt;br&gt; <code>UPDATE libref.member-name.ACCESS;</code> &lt;br&gt; <code>database-connection-statements;</code> &lt;br&gt; <code>editing-statements;</code> &lt;br&gt; <code>RUN;</code></td>
</tr>
</tbody>
</table>
### Task | Statements and Options to Use
---|---
Update an access descriptor and a view descriptor | **PROC ACCESS** statement-options;  
**UPDATE** libref.member-name.ACCESS;  
database-connection-statements;  
editing-statements;  
**UPDATE** libref.member-name.VIEW;  
editing-statements;  
**RUN**;

Update an access descriptor and create a view descriptor | **PROC ACCESS** statement-options;  
**UPDATE** libref.member-name.ACCESS;  
database-connection-statements;  
editing-statements;  
**CREATE** libref.member-name.VIEW;  
**SELECT** column-list;  
editing-statements;  
**RUN**;

Update a view descriptor from an existing access descriptor | **PROC ACCESS** statement-options, including ACCDESC=libref.access-descriptor;  
**UPDATE** libref.member-name.VIEW;  
editing-statements;  
**RUN**;

Create a SAS data set from a view descriptor | **PROC ACCESS** statement-options, including DBMS=dbms-name;  
VIEWDESC=libref.member; OUT=libref.member;  
**RUN**;

---

### DBMS Specifics: ACCESS Procedure

These SAS/ACCESS interfaces support the ACCESS procedure:

- **DB2 z/OS**
- **Oracle**
- **Sybase**

---

### Syntax: ACCESS Procedure

**See:**  
[DB2 z/OS](#), [Oracle](#), [Sybase](#)
PROC ACCESS <options>

database-connection-statements;

CREATE libref.member-name.ACCESS | VIEW <password-option>

UPDATE libref.member-name.ACCESS | VIEW <password-option>

TABLE= '<table-name>'

ASSIGN <= YES | NO | Y | N

DROP <>'column-identifier-1' <...'column-identifier-n'>'

FORMAT <>'column-identifier-1' => SAS-format-name-1
<...'column-identifier-n' => SAS-format-name-n>

LIST <ALL | VIEW | '<column-identifier>'>

QUIT;

RENAME <>'column-identifier-1' <= SAS-variable-name-1 <...'column-identifier-n' <= SAS-variable-name-n>

RESET ALL | <>'column-identifier-1' <...'column-identifier-n'>'

SELECT ALL | <>'column-identifier-1' <...'column-identifier-n' '

SUBSET selection-criteria;

UNIQUE <= YES | NO | Y | N

RUN;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC ACCESS</td>
<td>Access relational DBMS data</td>
</tr>
<tr>
<td>Database Connection</td>
<td>Provide DBMS-specific connection information</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>Indicate whether SAS variable names and formats are generated</td>
</tr>
<tr>
<td>CREATE</td>
<td>Create a SAS/ACCESS descriptor file</td>
</tr>
<tr>
<td>DROP</td>
<td>Drop a column so that it cannot be selected in a view descriptor</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Change a SAS format for a DBMS column</td>
</tr>
<tr>
<td>LIST</td>
<td>List columns in the descriptor and give information about them</td>
</tr>
<tr>
<td>QUIT</td>
<td>Terminate the procedure</td>
</tr>
<tr>
<td>RENAME</td>
<td>Modify the SAS variable name</td>
</tr>
<tr>
<td>RESET</td>
<td>Reset DBMS columns to their default settings</td>
</tr>
<tr>
<td>SELECT</td>
<td>Select DBMS columns for the view descriptor</td>
</tr>
<tr>
<td>SUBSET</td>
<td>Add or modify selection criteria for a view descriptor</td>
</tr>
<tr>
<td>TABLE=</td>
<td>Identify the DBMS table on which the access descriptor is based</td>
</tr>
<tr>
<td>UNIQUE</td>
<td>Generate SAS variable names based on DBMS column names</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Update a SAS/ACCESS descriptor file</td>
</tr>
</tbody>
</table>
PROC ACCESS Statement

Accesses relational DBMS data

Syntax

PROC ACCESS <options>;

Optional Arguments

ACCDESC=libref.access-descriptor
specifies an access descriptor. ACCDESC= is used with the DBMS= option to create or update a view descriptor that is based on the specified access descriptor. You can use a SAS data set option on the ACCDESC= option to specify any passwords that have been assigned to the access descriptor.

Note: The ODBC interface does not support this option.

DBMS=database-management-system
specifies which database management system you want to use. This DBMS-specific option is required. For details, see the DBMS-specific information for your SAS/ACCESS interface.

OUT=libref.member-name
specifies the SAS data file to which DBMS data is output.

VIEWDESC=libref.view-descriptor
specifies a view descriptor through which you extract the DBMS data.

Database Connection Statement

Provides DBMS-specific connection information

See: Database connection statements are used to connect to your DBMS. For details, see the DBMS-specific information for your SAS/ACCESS interface.

Syntax

database-connection-statements;

ASSIGN Statement

Indicates whether SAS variable names and formats are generated

Default: NO

Interactions: FORMAT, RENAME, RESET, UNIQUE
applies to access descriptor
Syntax
ASSIGN <=& YES | NO | Y | N;

Required Arguments
YES
generates unique SAS variable names from the first eight characters of the DBMS column names. If you specify YES, you cannot specify the RENAME, FORMAT, RESET, or UNIQUE statements when you create view descriptors that are based on the access descriptor.

NO
lets you modify SAS variable names and formats when you create an access descriptor and when you create view descriptors that are based on this access descriptor.

Details
The ASSIGN statement indicates how SAS variable names and formats are assigned:
• SAS automatically generates SAS variable names.
• You can change SAS variable names and formats in the view descriptors that are created from the access descriptor.

Each time the SAS/ACCESS interface encounters a CREATE statement to create an access descriptor, the ASSIGN statement is reset to the default NO value.

When you create an access descriptor, use the RENAME statement to change SAS variable names and the FORMAT statement to change SAS formats.

When you specify YES, SAS generates names according to these rules:
• You can change the SAS variable names only in the access descriptor.
• SAS variable names that are saved in an access descriptor are always used when view descriptors are created from the access descriptor. You cannot change them in the view descriptors.
• The ACCESS procedure allows names only up to eight characters.

CREATE Statement
Creates a SAS/ACCESS descriptor file

Requirement: This statement is required.
Interaction: applies to access descriptor or view descriptor
See: Statement Sequence for Accomplishing Tasks on page 956

Syntax
CREATE libref.member-name.ACCESS | VIEW <password-option>;
**Required Arguments**

*libref.*member-name

identifies the libref of the SAS library where you want to store the descriptor and identifies the descriptor name.

**ACCESS**

specifies an access descriptor.

**VIEW**

specifies a view descriptor.

**Optional Argument**

*password-option*

specifies a password.

**Details**

This statement names the access descriptor or view descriptor that you are creating. Use a three-level name:

- The first level identifies the libref of the SAS library where you want to store the descriptor.
- The second level is the descriptor name.
- The third level specifies the type of SAS file (specify **ACCESS** for an access descriptor or **VIEW** for a view descriptor).

**Example**

Within the same PROC ACCESS step, this example creates an AdLib.Employ access descriptor on the Employees Oracle table and a Vlib.Emp1204 view descriptor based on AdLib.Employ.

```sas
proc access dbms=oracle;
/* create access descriptor */
create adlib.employ.access;
database='qa:[dubois]textile';
table=employees;
assign=no;
list all;

/* create view descriptor */
create vlib.emp1204.view;
select empid lastname hiredate salary dept
gender birthdate;
format empid 6.
salary dollar12.2
jobcode 5.
hiredate datetime9.
birthdate datetime9.;
subset where jobcode=1204;
run;
```
**DROP Statement**

Drops a column so that it cannot be selected in a view descriptor

**Interactions:**
RESET, SELECT

applies to access or view descriptors

**Syntax**

DROP <column-identifier-1> <…<column-identifier-n>>;

**Required Argument**

*column-identifier*

specifies the column name or the positional equivalent from the LIST statement. This is the number that represents the column's place in the access descriptor. For example, to drop the third and fifth columns, submit this statement: **drop 3 5;**

**Details**

The DROP statement drops one or more specified columns from a descriptor. You can drop a column when you create or update an access descriptor. You can also drop a column when you update a view descriptor. If you drop a column when you create an access descriptor, you cannot select that column when you create a view descriptor that is based on the access descriptor. This statement does not affect the underlying DBMS table.

To display a previously dropped column, specify that column name in the RESET statement. However, doing this also resets all column attributes, such as the SAS variable name and format, to their default values.

---

**FORMAT Statement**

Changes a SAS format for a DBMS column

**Interactions:**
ASSIGN, DROP, RESET

applies to access or view descriptors

**Syntax**

FORMAT <column-identifier-1>
<=>SAS-format-name-1 <…<column-identifier-n> => SAS-format-name-n>;

**Required Arguments**

*column-identifier*

specifies the column name or the positional equivalent from the LIST statement. This is the number that represents the column's place in the access descriptor. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks.
**SAS-format-name**

specifies the SAS format to be used.

**Details**

The FORMAT statement changes SAS variable formats from their default formats. The default SAS variable format is based on the data type of the DBMS column. For details about default formats that SAS assigns to your DBMS data types, see the DBMS-specific information for your SAS/ACCESS interface.

You can use the FORMAT statement with a view descriptor only if the ASSIGN statement that was used when creating the access descriptor was specified with the **NO** value. When you use the FORMAT statement with access descriptors, the FORMAT statement also reselects columns that were previously dropped with the DROP statement.

For example, submit this statement to associate the DATE9. format with the BIRTHDATE column and with the second column in the access descriptor:

```sas
format 2=date9. birthdate=date9.;
```

The equal sign (=) is optional. For example, you can use the FORMAT statement to specify new SAS variable formats for four DBMS table columns:

```sas
format productid  4.
    weight     e16.9
    fibersize  e20.13
    width      e16.9;
```

**LIST Statement**

Lists columns in the descriptor and gives information about them

**Default:** ALL

**Restrictions:** You can use this statement only when you create an access descriptor or a view descriptor.

Changes that were made since the last CREATE, UPDATE, or RUN statement are not saved. Changes are saved only when a new CREATE, UPDATE, or RUN statement is submitted.

**Interactions:** applies to access or view descriptors

To review the contents of an existing view descriptor, use the CONTENTS procedure.

**Notes:** LIST information is written to your SAS log.

When you use LIST for an access descriptor, *NON-DISPLAY* appears next to the column description for any column that has been dropped and *UNSUPPORTED* appears next to any column if your DBMS interface view engine does not support the data type. When you use LIST for a view descriptor, *SELECTED* appears next to the column description for columns that you have selected for the view.

**Tip:** Specify LIST last in your PROC ACCESS code to see the entire descriptor. If you create or update multiple descriptors, specify LIST before each CREATE or UPDATE statement to list information about all descriptors that you are creating or updating.
Syntax

LIST <ALL | VIEW | 'column-identifier'>;

Optional Arguments

ALL
lists all DBMS columns in the table, positional equivalents, SAS variable names, and SAS variable formats that are available for a descriptor.

VIEW
lists all DBMS columns that are selected for a view descriptor, their positional equivalents, their SAS names and formats, and any subsetting clauses.

column-identifier
lists information about a specified DBMS column, including its name, positional equivalent, SAS variable name and format, and whether it has been selected. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks.

The column-identifier argument can be either the column name or the positional equivalent. This is the number that represents the column's place in the descriptor. For example, to list information about the fifth column in the descriptor, submit this statement:

list 5;

QUIT Statement
Terminates the procedure without any further descriptor creation

Interaction: applies to access or view descriptors

Syntax

QUIT;

RENAME Statement
Sets or modifies the SAS variable name that is associated with a DBMS column

Interactions: ASSIGN, RESET
applies to access or view descriptors

Syntax

RENAME 'column-identifier-1' => SAS-variable-name-1 <…>'column-identifier-n' => SAS-variable-name-n>;

Required Arguments

column-identifier
specifies the DBMS column name or the positional equivalent from the LIST statement. This is the number that represents the column's place in the descriptor. If
the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks. The equal sign (=) is optional.

*SAS-variable-name*

specifies a SAS variable name.

**Details**

Two factors affect the use of the RENAME statement: whether you specify the ASSIGN statement when you are creating an access descriptor, and the type of descriptor that you are creating.

- If you omit the ASSIGN statement or specify it with a NO value, the renamed SAS variable names that you specify in the access descriptor are retained when an ACCESS procedure executes. For example, if you rename the CUSTOMER column to CUSTNUM when you create an access descriptor, the column is still named CUSTNUM when you select it in a view descriptor. The exception would be if you specified another RESET or RENAME statement.

  When you create a view descriptor that is based on this access descriptor, you can specify the RESET statement or another RENAME statement to rename the variable. However, the new name applies only in that view. When you create other view descriptors, the SAS variable names are derived from the access descriptor.

- If you specify the YES value in the ASSIGN statement, you can use the RENAME statement to change SAS variable names only while creating an access descriptor. As described earlier, SAS variable names and formats that are saved in an access descriptor are always used when creating view descriptors that are based on the access descriptor.

For example, to rename the SAS variable names that are associated with the seventh column and the nine-character FIRSTNAME column in a descriptor, submit this statement:

```plaintext
rename
  7 birthdy  'firstname'=fname;
```

When you are creating a view descriptor, the RENAME statement automatically selects the renamed column for the view. That is, if you rename the SAS variable associated with a DBMS column, you do not have to issue a SELECT statement for that column.

**RESET Statement**

Resets DBMS columns to their default settings

**Interactions:** ASSIGN, DROP, FORMAT, RENAME, SELECT

**Syntax**

```plaintext
RESET ALL | '<column-identifier-1>' <...>'column-identifier-n'>';
```
Required Arguments

ALL

resets all columns in an access descriptor to their default names and formats and reselects any dropped columns. ALL deselects all columns in a view descriptor so that no columns are selected for the view.

column-identifier

can be either the DBMS column name or the positional equivalent from the LIST statement. This is the number that represents the column's place in the access descriptor. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks. For example, to reset the SAS variable name and format associated with the third column, submit this statement: reset 3;

For access descriptors, the specified column is reset to its default name and format settings. For view descriptors, the specified column is no longer selected for the view.

Details

This statement has different effects on access and view descriptors.

For access descriptors, the RESET statement resets the specified column names to the default names that are generated by the ACCESS procedure. The RESET statement also changes the current SAS variable format to the default SAS format. Any previously dropped columns that are specified in the RESET statement become available.

When creating an access descriptor, if you omit the ASSIGN statement or set it to NO, the default SAS variable names are blanks. If you set ASSIGN=YES, default names are the first eight characters of each DBMS column name.

For view descriptors, the RESET statement clears (deselects) any columns that were included in the SELECT statement. When you create a view descriptor that is based on an access descriptor that is created without an ASSIGN statement or with ASSIGN=NO, resetting and then reselecting (within the same procedure execution) a SAS variable changes the SAS variable names and formats to their default values. When you create a view descriptor that is based on an access descriptor created with ASSIGN=YES, the RESET statement does not have this effect.

SELECT Statement

Selects DBMS columns for the view descriptor

Requirement: This statement is required.

Interactions:

RESET

applies to view descriptors

Note: The SELECT statement specifies which DBMS columns in an access descriptor to include in a view descriptor.

Tip: To clear your current selections when creating a view descriptor, use the RESET ALL statement.

Syntax

SELECT ALL | <>column-identifier-1<> | ... | <>column-identifier-n<>;
**Required Arguments**

**ALL**
includes in the view descriptor all columns that were defined in the access descriptor and that were not dropped.

**column-identifier**
can be either the DBMS column name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents where the column is located in the access descriptor on which the view is based. For example, to select the first three columns, submit this statement:

```plaintext
select 1 2 3;
```

If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotation marks.

**Details**

SELECT statements are cumulative within a view creation. For example, columns 1, 5, and 6 are selected if you submit these SELECT statements:

```plaintext
select 1;
select 5 6;
```

**SUBSET Statement**

Adds or modifies selection criteria for a view descriptor

**Restriction:**
The SUBSET statement is case sensitive, unlike other ACCESS procedure statements. The SQL statement is sent to the DBMS exactly as you enter it. You must therefore use the correct case for any DBMS object names. For details, see the DBMS-specific information for your SAS/ACCESS interface.

**Interactions:**
apply to view descriptors

If you specify more than one SUBSET statement per view descriptor, the last SUBSET overwrites the earlier SUBSETs.

**Notes:**
This statement is optional.

SAS does not check the SUBSET statement for errors. The statement is verified only when the view descriptor is used in a SAS program.

**Tips:**
You can use the SUBSET statement to specify selection criteria when you create a view descriptor.

If you omit it, the view retrieves all data (rows) in the DBMS table.

To delete the selection criteria, submit a SUBSET statement without any arguments.

**Example:**
For a view descriptor that retrieves rows from a DBMS table, you could submit this statement:

```plaintext
subset where firstorder is not null;
```

**Syntax**

```plaintext
SUBSET selection-criteria;
```
**Required Argument**

*selection-criteria*

one or more DBMS-specific SQL expressions that are accepted by your DBMS, such as WHERE, ORDER BY, HAVING, and GROUP BY. Use DBMS column names, not SAS variable names, in your selection criteria.

**Details**

If you have multiple selection criteria, enter them all in one SUBSET statement, as shown in this example:

```
subset where firstorder is not null
    and country = 'USA'
order by country;
```

---

**TABLE= Statement**

Identifies the DBMS table on which the access descriptor is based

**Interactions:**

- applies to access descriptors
- required with the CREATE statement
- optional with the UPDATE statement

**Syntax**

```
TABLE= '<>'table-name<>';
```

**Required Argument**

*table-name*

a valid DBMS table name. If it contains lowercase characters, special characters, or national characters, you must enclose it in quotation marks. For details, see the DBMS-specific information for your SAS/ACCESS interface.

---

**UNIQUE Statement**

Generates SAS variable names based on DBMS column names

**Interactions:**

ASSIGN

- applies to view descriptors

**Notes:**

- It is recommended that you use the UNIQUE statement and specify UNIQUE=YES. If you omit the UNIQUE statement or specify UNIQUE=NO and SAS encounters duplicate SAS variable names in a view descriptor, your job fails.
- The equal sign (=) is optional in the UNIQUE statement.

**Tip:**

If duplicate SAS variable names exist in the access descriptor on which you are creating a view descriptor, you can specify UNIQUE to resolve the duplication.

**See:**

RENAME statement
Syntax

UNIQUE <= Y | N | NO | YES;

Required Arguments

YES
causes the SAS/ACCESS interface to append numbers to any duplicate SAS variable names, making each variable name unique.

NO
causes the SAS/ACCESS interface to continue to allow duplicate SAS variable names to exist. You must resolve these duplicate names before saving (and thereby creating) the view descriptor.

Details

The UNIQUE statement specifies whether the SAS/ACCESS interface should generate unique SAS variable names for DBMS columns for which SAS variable names have not been entered.

The UNIQUE statement is affected by whether you specified the ASSIGN statement when you created the access descriptor on which the view is based:

• If you specified the ASSIGN=YES statement, you cannot specify UNIQUE when creating a view descriptor. YES causes SAS to generate unique names, so UNIQUE is not necessary.

• If you omitted the ASSIGN statement or specified ASSIGN=NO, you must resolve any duplicate SAS variable names in the view descriptor. You can use UNIQUE to generate unique names automatically, or you can use the RENAME statement to resolve duplicate names yourself.

UPDATE Statement

Updates a SAS/ACCESS descriptor file

Interactions: applies to access or view descriptors
ASSIGN, RESET, SELECT, UNIQUE statements are not supported when you use the UPDATE statement.

Note: Rules that apply to the CREATE statement also apply to the UPDATE statement. For example, the SUBSET statement is valid only for updating view descriptors.

See: Statement Sequence for Accomplishing Tasks on page 956

Syntax

UPDATE libref.member-name.ACCESS | VIEW <password-option>;

Required Arguments

libref.member-name
identifies the libref of the SAS library where you want to store the descriptor and identifies the descriptor name.
ACCESS
   specifies an access descriptor.

VIEW
   specifies a view descriptor.

Optional Argument

password-option
   specifies a password.

Details

The UPDATE statement identifies an existing access descriptor or view descriptor that you want to update. UPDATE is normally used to update database connection information, such as user IDs and passwords. If your descriptor requires many changes, it might be easier to use the CREATE statement to overwrite the old descriptor with a new one.

Altering a DBMS table might invalidate descriptor files that are based on the DBMS table, or it might cause these files to be out of date. If you re-create a table, add a new column to a table, or delete an existing column from a table, use the UPDATE statement to modify your descriptors so that they use the new information.

Using Descriptors with the ACCESS Procedure

What Are Descriptors?

Descriptors work with the ACCESS procedure by providing information about DBMS objects to SAS, enabling you to access and update DBMS data from within a SAS session or program.

There are two types of descriptors, access descriptors and view descriptors. Access descriptors provide SAS with information about the structure and attributes of a DBMS table or view. In turn, an access descriptor is used to create one or more view descriptors, or SAS data views, of the DBMS data.

Access Descriptors

Typically, each DBMS table or view has a single access descriptor that provides connection information, data type information, and names for databases, tables, and columns.

You use an access descriptor to create one or more view descriptors. When creating a view descriptor, you select the columns and specify criteria for the rows that you want to retrieve. The figure below illustrates the descriptor creation process. Note that an access descriptor, which contains the metadata of the DBMS table, must be created before view descriptors can be created.
**View Descriptors**

You use a view descriptor in a SAS program much as you would any SAS data set. For example, you can specify a view descriptor in the DATA= statement of a SAS procedure or in the SET statement of a DATA step.

You can also use a view descriptor to copy DBMS data into a SAS data file, called extracting the data. When you need to use DBMS data in several procedures or DATA steps, you might use fewer resources by extracting the data into a SAS data file instead of repeatedly accessing the data directly.

The SAS/ACCESS interface view engine usually tries to pass WHERE conditions to the DBMS for processing. In most cases it is more efficient for a DBMS to process WHERE conditions than for SAS to do the processing.

**Accessing Data Sets and Descriptors**

SAS lets you control access to SAS data sets and access descriptors by associating one or more SAS passwords with them. When you create an access descriptor, the connection information that you provide is stored in the access descriptor and in any view descriptors based on that access descriptor. The password is stored in an encrypted form. When these descriptors are accessed, the connection information that was stored is also used to access the DBMS table or view. To ensure data security, you might want to change the protection on the descriptors to prevent others from seeing the connection information stored in the descriptors.

When you create or update view descriptors, you can use a SAS data set option after the ACCDESC= option to specify the access descriptor password, if one exists. In this case,
you are not assigning a password to the view descriptor that is being created or updated. Instead, using the password grants you permission to use the access descriptor to create or update the view descriptor. Here is an example:

```sas
proc access dbms=sybase accdesc=adlib.customer {alter=rouge};
    create vlib.customer.view;
    select all;
run;
```

By specifying the ALTER level of password, you can read the AdLib.Customer access descriptor and create the Vlib.Customer view descriptor.

### Examples: ACCESS Procedure

#### Example 1: Update an Access Descriptor

**Features:**
- Statements
  - PROC ACCESS
  - UPDATE
  - DROP
  - LIST

The following example updates an AdLib.Employ access descriptor on the Employees table in Oracle. The original access descriptor includes all columns in the table. The updated access descriptor omits the Salary and BirthDate columns. You can use the LIST statement to write all variables to the SAS log so that you can see the complete access descriptor before you update it.

```sas
proc access dbms=oracle ad=adlaib.employ;
    /* update access descriptor */
    update adlib.employ.access;
    drop salary birthdate;
    list all;
run;
```

#### Example 2: Create a View Descriptor

**Features:**
- Statements
  - PROC ACCESS
  - CREATE
  - SELECT
  - FORMAT
  - SUBSET

Because SELECT and RESET are not supported when UPDATE is used, the view descriptor Vlib.Emp1204 must be re-created to omit the Salary and BirthDate columns. Based on a previously updated access descriptor, ADLIB.EMPLOY, the following example re-creates the VLIB.EMP1204 view descriptor.

```sas
proc access dbms=oracle;
```
/* re-create view descriptor */
create vlib.emp1204.view;
select empid hiredate dept jobcode gender
   lastname firstname middlename phone;
format empid 6.
   jobcode 5.
   hiredate datetime9.;
subset where jobcode=1204;
run;
Overview: CV2VIEW Procedure

SAS still supports this legacy procedure, which converts into SQL views any SAS/ACCESS 32- and 64-bit view descriptors that were created prior to SAS 9.2. To access your relational DBMS data more directly, it is recommended that you convert your descriptors so that you can instead use the preferred method, the SAS/ACCESS LIBNAME statement on page 96. The LIBNAME statement provides greater control over DBMS operations such as locking, spooling, and data type conversions. It can handle long field names, which descriptors cannot. Also, SQL views are platform-independent, whereas descriptors are not.

If the descriptor that you want to convert is READ-, WRITE-, or ALTER-protected, those values are applied to the output SQL view. For security reasons, these values do not appear if you save the generated SQL to a file. The PASSWORD part of the LIBNAME statement is also not visible to prevent generated SQL statements from being submitted manually without modification.
PROC CV2VIEW Statement
Converts SAS/ACCESS view descriptors into SQL views

**Syntax**

PROC CV2VIEW DBMS=\textit{dbms-name} | ALL;

**Statement** | **Task**
--- | ---
PROC CV2VIEW | Convert SAS/ACCESS view descriptors into SQL views
FROM_LIBREF= | Specify the library containing the view descriptors or access descriptors that you want to convert
FROM_VIEW= | Specify the view descriptor or access descriptor that you want to convert
REPLACE= | Specify whether existing views and files are replaced
SAVEAS= | Save the generated PROC SQL statements to a file
SUBMIT | Submit the generated PROC SQL statements when you specify the SAVEAS= statement
TO_VIEW= | Specify the name of the new (converted) SQL view
TO_LIBREF= | Specify the library that contains the new (converted) SQL views
TYPE= | Specify what type of conversion should occur
**Required Arguments**

`dbms-name`

specifies the name of a supported database from which you want to obtain descriptors. Valid values for `dbms-name` are `DB2`, `Oracle`, and `Sybase`.

`ALL`

specifies that you want the descriptors from all supported databases.

---

**FROM_VIEW= Statement**

Specifies the name of the view descriptor or access descriptor that you want to convert

**Restriction:**
If you specify `DBMS=ALL`, you cannot use the `FROM_VIEW=` statement.

**Requirements:**
You must specify either the `FROM_VIEW=` statement or the `FROM_LIBREF=` statement.

FROM_VIEW= and TO_VIEW= statements are always used together.

**Syntax**

`FROM_VIEW=libref.input-descriptor;`

**Required Arguments**

`libref`

specifies the libref that contains the view descriptor or access descriptor that you want to convert.

`input-descriptor`

specifies the view descriptor or access descriptor that you want to convert.

---

**FROM_LIBREF= Statement**

Specifies the library that contains the view descriptors or access descriptors that you want to convert

**Requirements:**
You must specify either the `FROM_VIEW=` statement or the `FROM_LIBREF=` statement.

FROM_LIBREF= and TO_LIBREF= statements are always used together.

**Syntax**

`FROM_LIBREF=input-library;`

**Required Argument**

`input-library`

specifies a previously assigned library that contains the view descriptors or access descriptors that you want to convert. All descriptors that are in the specified library and that access data in the specified DBMS are converted into SQL views. If you specify `DBMS=ALL`, all descriptors that are in the specified library and that access any supported DBMS are converted.
REPLACE= Statement
Specifies whether existing views and files are replaced

Syntax
REPLACE=ALL | FILE | VIEW;

Required Arguments
ALL
replaces the TO_VIEW= file if it already exists and replaces the SAVEAS= file if it already exists.

FILE
replaces the SAVEAS= file if it already exists. If the file already exists, and if REPLACE=FILE or REPLACE=ALL is not specified, the generated PROC SQL code is appended to the file.

VIEW
replaces the TO_VIEW= file if it already exists.

SAVEAS= Statement
Saves the generated PROC SQL statements to a file

Interaction: If you specify the SAVEAS= statement, the generated SQL is not automatically submitted, so you must use the SUBMIT statement on page 979.

Syntax
SAVEAS=external-filename;

Required Argument
external-filename
lets you save the PROC SQL statements that are generated by PROC CV2VIEW to an external file. You can modify this file and submit it on another platform.

Details
PROC CV2VIEW inserts comments in the generated SQL to replace any statements that contain passwords. For example, if a view descriptor is READ-, WRITE-, or ALTER-protected, the output view has the same level of security. However, the file that contains the SQL statements does not show password values. The password in the LIBNAME statement also does not show password values.
**SUBMIT Statement**
Causes PROC CV2VIEW to submit the generated PROC SQL statements when you specify the SAVEAS= statement

**Tip:** If you do not use the SAVEAS= statement on page 978, PROC CV2VIEW automatically submits the generated SQL. So you do not need to specify the SUBMIT statement.

**Syntax**

```
SUBMIT;
```

**TO_VIEW= Statement**
Specifies the name of the new (converted) SQL view

**Restriction:** If you specify DBMS=ALL, you cannot use the TO_VIEW= statement.

**Requirements:** You must specify either the TO_VIEW= statement or the TO_LIBREF= statement. FROM_VIEW= and TO_VIEW= statements are always used together.

**Interaction:** Use the REPLACE= statement on page 978 to control whether the output file is overwritten or appended if it already exists.

**Syntax**

```
TO_VIEW=libref.output-view;
```

**Required Arguments**

**libref**
specifies the libref where you want to store the new SQL view.

**output-view**
specifies the name for the new SQL view that you want to create.

**TO_LIBREF= Statement**
Specifies the library that contains the new (converted) SQL views

**Requirements:** You must specify either the TO_VIEW= statement or the TO_LIBREF= statement. The FROM_LIBREF= and TO_LIBREF= statements are always used together.

**Interaction:** Use the REPLACE= statement on page 978 if a file with the name of one of your output views already exists. If a file with the name of one of your output views already exists and you do not specify the REPLACE statement, PROC CV2VIEW does not convert that view.
Syntax

TO_LIBREF= output-library;

Required Argument

output-library
specifies the name of a previously assigned library where you want to store the new SQL views.

Details

The names of the input view descriptors or access descriptors are used as the output view names. In order to individually name your output views, use the FROM_VIEW= statement on page 977 and the TO_VIEW= statement on page 979.

TYPE= Statement

Specifies what type of conversion should occur

Syntax

TYPE=SQL | VIEW | ACCESS;

Required Arguments

SQL
specifies that PROC CV2VIEW converts descriptors to SQL views. This is the default behavior.

VIEW
specifies that PROC CV2VIEW converts descriptors to native view descriptor format. It is most useful in the 32-bit to 64-bit case. It does not convert view descriptors across different operating systems.

ACCESS
specifies that PROC CV2VIEW converts access descriptors to native access descriptor format. It is most useful in the 32-bit to 64-bit case. It does not convert access descriptors across different operating systems.

Details

If TYPE=VIEW or TYPE=ACCESS, then SAVEAS=, SUBMIT, and REPLACE= or REPLACE_FILE= are not valid options.

Examples: CV2VIEW Procedure

Example 1: Converting an Individual View Descriptor

Features: Statements
PROC CV2VIEW
FROM_VIEW=
TO_VIEW=
SAVEAS=
SUBMIT
REPLACE

In this example, PROC CV2VIEW converts the MYVIEW view descriptor to the SQL view NEWVIEW. When you use ALTER, READ, and WRITE, the MYVIEW view descriptor is protected against alteration, reading, and writing. The PROC SQL statements that PROC CV2VIEW generates are submitted and saved to an external file named SQL.SAS.

Details

The REPLACE FILE statement causes an existing file named SQL.SAS to be overwritten. Without this statement, the text would be appended to SQL.SAS if the user has the appropriate privileges.

The LABEL value of EMPINFO is the name of the underlying database table that is referenced by the view descriptor.

If the underlying DBMS is Oracle or DB2, the CV2VIEW procedure adds the PRESERVE_TAB_NAMES= option to the embedded LIBNAME statement. You can then use CV2VIEW to access those tables with mixed-case or embedded-blank table names.

Note: This SQL syntax fails if you try to submit it because the PW field of the LIBNAME statement is replaced with a comment in order to protect the password. The ALTER, READ, and WRITE protection is commented out for the same reason. You can add the passwords to the code and then submit the SQL to re-create the view.

libname input '/username/descriptors/';
libname output '/username/sqlviews/';

proc cv2view dbms=oracle;
from_view = input.myview (alter=apwd);
to_view = output.newview;
saveas = '/username/vsql/sql.sas';
submit;
replace file;
run;

PROC CV2VIEW generates these PROC SQL statements.

 cuerpo de la consulta de SQL
Example 2: Converting a Library of View Descriptors for a Single DBMS

In this example PROC CV2VIEW converts all Oracle view descriptors in the input library into SQL views. If an error occurs during the conversion of a view descriptor, the procedure moves to the next view. The PROC SQL statements that PROC CV2VIEW generates are both submitted and saved to an external file named SQL.SAS.

The SAVEAS= statement causes all generated SQL for all Oracle view descriptors to be stored in the MANYVIEW.SAS file.

If the underlying DBMS is Oracle or DB2, the CV2VIEW procedure adds the PRESERVE_TAB_NAMES= option to the embedded LIBNAME statement. You can then use CV2VIEW to access those tables with mixed-case or embedded-blank table names.

PROC CV2VIEW generates these PROC SQL statements for one of the views.

```sql
/* SOURCE DESCRIPTOR: PPCV2R */
PROC SQL DQUOTE=ANSI;
  CREATE VIEW OUTPUT.PPCV2R
  (LABEL=EMPLOYEES
  )
```
Example 3: Converting a Library of View Descriptors for All Supported DBMSs

Features:

PROC CV2VIEW option
  DBMS=ALL

Statements

PROC CV2VIEW
  FROM_LIBREF=
  TO_LIBREF=

In this example PROC CV2VIEW converts all view descriptors that are in the input library and that access data in any supported DBMS. If an error occurs during the conversion of a view descriptor, the procedure then moves to the next view. The PROC SQL statements that PROC CV2VIEW generates are automatically submitted but are not saved to an external file because the SAVEAS= statement on page 978 is not used.

libname input '/username/descriptors';
libname output '/username/sqlviews';
proc cv2view dbms=all;
from_libref = input;
to_libref = output;
run;
Overview: DBLOAD Procedure

Sending Data from SAS to a DBMS

SAS still supports this legacy procedure. However, to send data from SAS to a DBMS more directly, it is recommended that you use the SAS/ACCESS LIBNAME statement on page 96 or the SQL pass-through facility on page 472. To determine whether this feature is available in your environment for your interface, see SAS/ACCESS Features by Host on page 77.
Properties of the DBLOAD Procedure

Along with the ACCESS procedure and an interface view engine, the DBLOAD procedure creates an interface between SAS and data in other vendors' databases.

The DBLOAD procedure enables you to create and load a DBMS table, append rows to an existing table, and submit non-query DBMS-specific SQL statements to the DBMS for processing. The procedure constructs DBMS-specific SQL statements to create and load, or append, to a DBMS table by using one of these items:

• a SAS data file
• an SQL view or DATA step view
• a view descriptor that was created with the SAS/ACCESS interface to your DBMS or with another SAS/ACCESS interface product
• another DBMS table referenced by a SAS libref that was created with the SAS/ACCESS LIBNAME statement.

The DBLOAD procedure associates each SAS variable with a DBMS column and assigns a default name and data type to each column. It also specifies whether each column accepts NULL values. You can use the default information or change it as necessary. When you are finished customizing the columns, the procedure creates the DBMS table and loads or appends the input data.

About DBLOAD Procedure Statements

This procedure has several types of statements:

• database connection statements for connecting to your DBMS (see DBMS-specific information in this document for your SAS/ACCESS interface)

• creating and loading statements: LOAD and RUN

• table and editing statements: ACCDESC=, COMMIT=, DELETE, ERRLIMIT=, LABEL, LIMIT=, LIST, NULLS, QUIT, RENAME, RESET, SQL, TABLE=, TYPE, WHERE

This table summarizes PROC DBLOAD options and statements that are required to accomplish common tasks.

Table A3.1 Statement Sequence for Accomplishing Common Tasks with the DBLOAD Procedure

<table>
<thead>
<tr>
<th>Task</th>
<th>Options and Statements to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and load a DBMS table (LOAD must appear before RUN to create and load a table or append data to a table)</td>
<td>PROC DBLOAD statement-options; database-connection-options;</td>
</tr>
<tr>
<td></td>
<td>TABLE= '&lt;table-name&gt;';</td>
</tr>
<tr>
<td></td>
<td>LOAD;</td>
</tr>
<tr>
<td></td>
<td>RUN;</td>
</tr>
<tr>
<td>Task</td>
<td>Options and Statements to Use</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Submit a dynamic, non-query DBMS-SQL statement to DBMS (without creating a table) | **PROC DBLOAD**  
statement-options;  
database-connection-options;  
**SQL DBMS-specific-SQL-statements**;  
**RUN**;                                                                 |

**DBMS Specifics: DBLOAD Procedure**

These SAS/ACCESS interfaces support this procedure:

- DB2 under UNIX and PC Hosts
- DB2 under z/OS
- Microsoft SQL Server
- ODBC
- Oracle
- Sybase

**Syntax: DBLOAD Procedure**

See: DB2 under UNIX and PC Hosts, DB2 under z/OS, Microsoft SQL Server, ODBC, Oracle, Sybase
PROC DBLOAD <options>;

  database connection statements;
  TABLE= <'table-name'>;
  ACCDESC= <libref:access-descriptor>;
  COMMIT= commit-frequency;
  DELETE variable-identifier-1
    <...variable-identifier-n>;
  ERRLIMIT= error-limit;
  LABEL;
  LIMIT= load-limit;
  LIST <ALL | COLUMN | variable-identifier>;
  NULLS variable-identifier-1 = Y | N
    <...variable-identifier-n = Y | N>;
  QUIT;
  RENAME variable-identifier-1 = <'column-name-1'>
    <...variable-identifier-n = <'column-name-n'>>;
  RESET ALL | variable-identifier-1<...variable-identifier-n>;
  SQL DBMS-specific SQL-statement;
  TYPE variable-identifier-1 = 'column-type-1'
    <...variable-identifier-n = 'column-type-n'>;
  WHERE SAS-where-expression;
  LOAD;
  RUN;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC DBLOAD</td>
<td>Send data from SAS to a DBMS</td>
</tr>
<tr>
<td>Database</td>
<td>Provide DBMS connection information</td>
</tr>
<tr>
<td>ACCDESC=</td>
<td>Create an access descriptor based on the</td>
</tr>
<tr>
<td></td>
<td>new DBMS table</td>
</tr>
<tr>
<td>COMMIT=</td>
<td>Issue a commit or save rows after a</td>
</tr>
<tr>
<td></td>
<td>specified number of inserts</td>
</tr>
<tr>
<td>DELETE</td>
<td>Specify variables that should not be</td>
</tr>
<tr>
<td></td>
<td>loaded into the new table</td>
</tr>
<tr>
<td>ERRLIMIT=</td>
<td>Stop the loading of data after a</td>
</tr>
<tr>
<td></td>
<td>specified number of errors</td>
</tr>
<tr>
<td>LABEL</td>
<td>Cause DBMS column names to default to SAS</td>
</tr>
<tr>
<td>LIMIT=</td>
<td>Limit the number of observations that are</td>
</tr>
<tr>
<td>LIST</td>
<td>List information about the variables to</td>
</tr>
<tr>
<td>LOAD</td>
<td>Create and load the new DBMS table</td>
</tr>
<tr>
<td>NULLS</td>
<td>Specify whether DBMS columns accept NULL</td>
</tr>
<tr>
<td></td>
<td>values</td>
</tr>
<tr>
<td>QUIT</td>
<td>Terminate the procedure</td>
</tr>
<tr>
<td>RENAME</td>
<td>Rename DBMS columns</td>
</tr>
<tr>
<td>Statement</td>
<td>Task</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>RESET</td>
<td>Reset column names and data types to their default values</td>
</tr>
<tr>
<td>SQL</td>
<td>Submit a DBMS-specific SQL statement to the DBMS</td>
</tr>
<tr>
<td>TABLE=</td>
<td>Name the DBMS table to be created and loaded</td>
</tr>
<tr>
<td>TYPE</td>
<td>Change default DBMS data types in the new table</td>
</tr>
<tr>
<td>WHERE</td>
<td>Load a subset of data into the new table</td>
</tr>
</tbody>
</table>

**PROC DBLOAD Statement**

Sends data from SAS to a DBMS

**Syntax**

PROC DBLOAD <options>;

**Optional Arguments**

**DBMS=database-management-system**

specifies which database management system you want to access. This DBMS-specific option is required. See the DBMS-specific reference in this document for details about your DBMS.

**DATA=<libref.:SAS-data-set**

specifies the input data set. You can retrieve data from a SAS data file, an SQL view, a DATA step view, a SAS/ACCESS view descriptor, or another DBMS table to which a SAS/ACCESS libref points. If the SAS data set is permanent, you must use its two-level name, libref.SAS-data-set. If you omit the DATA= option, the default is the last SAS data set that was created.

**APPEND**

appends data to an existing DBMS table that you identify by using the TABLE= statement. When you specify APPEND, the input data that you specify with the DATA= option is inserted into the existing DBMS table. Your input data can be a SAS data set, SQL view, or SAS/ACCESS view (view descriptor).

**CAUTION:**

When you use APPEND, you must ensure that your input data corresponds exactly to the columns in the DBMS table. If your input data does not include values for all columns in the DBMS table, you might corrupt your DBMS table by inserting data into the wrong columns. Use the COMMIT, ERRLIMIT, and LIMIT statements to help safeguard against data corruption. Use the DELETE and RENAME statements to drop and rename SAS input variables that do not have corresponding DBMS columns.

You can use all PROC DBLOAD statements and options with APPEND except NULLS and TYPE, which have no effect. The LOAD statement is required.

The following example appends new employee data from the NEWEMP SAS data set to the EMPLOYEES DBMS table. The COMMIT statement causes a DBMS
commit to be issued after every 100 rows are inserted. The ERRLIMIT statement causes processing to stop after five errors occur.

```
proc dbload dbms=oracle data=newemp append;
  user=myusr1; password=mypwd1; path='mysrv1';
  table=employees; commit=100; errlimit=5; load;
run;
```

By omitting the APPEND option from the DBLOAD statement, you can use the PROC DBLOAD SQL statements to create a DBMS table and append to it in the same PROC DBLOAD step.

### Database Connection Statement

Provides DBMS connection information

**Interaction:** These statements are used to connect to your DBMS. They vary based on the SAS/ACCESS interface that you are using (for example, include USER=, PASSWORD=, and DATABASE=). See the DBMS-specific reference in this document for details about your SAS/ACCESS interface.

**Syntax**

```
database-connection-statements
```

### ACCDESC= Statement

Creates an access descriptor based on the new DBMS table that you are creating and loading

**Requirement:** You must specify an access descriptor if it does not already exist.  
**Tip:** If you specify ACCDESC=, the access descriptor is automatically created after the new table is created and loaded.

**Syntax**

```
ACCDESC=<libref:access-descriptor>;
```

### COMMIT= Statement

Issues a commit or saves rows after a specified number of inserts

**Default:** 1000  
**Requirement:** The commit-frequency argument must be a nonnegative integer.  
**Interaction:** If you omit the COMMIT= statement, a commit is issued or a group of rows is saved after each 1,000 rows are inserted and after the last row is inserted.  
**Note:** The COMMIT= statement issues a commit (generates a DBMS-specific SQL COMMIT statement) after the specified number of rows has been inserted.  
**Tip:** Using this statement might improve performance by releasing DBMS resources each time the specified number of rows has been inserted.
Syntax

COMMIT=commit-frequency;

DELETE Statement

drops the specified SAS variables before the DBMS table is created

Note: The variable-identifier argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable’s place in the data set. For example, if you want to drop the third variable, submit this statement: delete 3;

Tip: When you drop a variable, the positional equivalents of the variables do not change. For example, if you drop the second variable, the third variable is still referenced by the number 3, not 2. If you drop more than one variable, separate the identifiers with spaces, not commas.

Syntax

DELETE variable-identifier-1 <…variable-identifier-n>;

ERRLIMIT= Statement

Stops the loading of data after a specified number of errors

Default: 100 (see the DBMS-specific details for possible exceptions)

Requirement: The error-limit argument must be a nonnegative integer.

Interaction: The ERRLIMIT= statement defaults to 10 when used with APPEND.

Note: The ERRLIMIT= statement stops the loading of data after the specified number of DBMS SQL errors has occurred. Errors include observations that fail to be inserted and commits that fail to execute. If the SQL CREATE TABLE statement that the procedure generates fails, the procedure ends.

Tip: To allow an unlimited number of DBMS SQL errors to occur, specify ERRLIMIT=0.

Syntax

ERRLIMIT=error-limit;

LABEL Statement

Causes DBMS column names to default to SAS variable labels when the new table is created

Default: DBMS column names default to SAS variable names

Requirement: You must use the RESET statement after the LABEL statement for the LABEL statement to take effect.

Interaction: RESET
Note: If a SAS variable has no label, the variable name is used. If the label is too long to be a valid DBMS column name, the label is truncated.

Syntax

**LABEL;**

LIMIT= Statement

Limits the number of observations that can be loaded into the new DBMS table

**Default:** 5000

**Requirement:** The *load-limit* argument must be a nonnegative integer.

**Tip:** To load all observations from your input data set, specify LIMIT=0.

Syntax

**LIMIT=load-limit;**

LIST Statement

Lists information about some or all of the SAS variables to be loaded into the new DBMS table

**Default:** ALL

**Note:** By default, the list is sent to the SAS log.

**Tip:** You can specify LIST as many times as you want while creating a DBMS table. Specify LIST before the LOAD statement to see the entire table.

Syntax

**LIST <ALL | FIELD | variable-identifier>;**

*Optional Arguments*

**ALL**

lists information about all variables in the input SAS data set, despite whether those variables are selected for the load.

**FIELD**

lists information about only the input SAS variables that are selected for the load.

**variable-identifier**

lists information about only the specified variable. The *variable-identifier* argument can be either the SAS variable name or the positional equivalent. The positional equivalent is the number that represents the variable's position in the data set. For example, if you want to list information for the column associated with the third SAS variable, submit this statement: **list 3;**
LOAD Statement

Creates and loads the new DBMS table

**Restriction:** in the DBLOAD procedure (required statement for loading or appending data)

**Requirements:** This statement is required to create and load a new DBMS table or to append data to an existing table.

When you create and load a DBMS table, you must place statements or groups of statements in a certain order after the PROC DBLOAD statement and its options, as listed in Statement Sequence for Accomplishing Common Tasks on page 986.

**Note:** The LOAD statement informs the DBLOAD procedure to execute the action that you request, including loading or appending data.

**Example:** This example creates the SummerTemps table in Oracle based on the DLib.TempEmps data file.

```sas
proc dbload dbms=oracle data=dlib.tempemps;
   user=myusr1; password=mypwd1; path='mysrv1';
   table=summertemps; rename firstnam=firstname middlena=middlename;
   type hiredate 'date' empid 'number(6,0)' familyid 'number(6,0)';
   nulls 1=n; list; load;
run;
```

**Syntax**

LOAD;

NULLS Statement

Specifies whether DBMS columns accept NULL values

**Default:** Y

**Interaction:** Some DBMSs have three valid values for this statement, Y, N, and D. See the DBMS-specific reference in this document for details about your DBMS.

**Tips:** The `variable-identifier` argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable’s place in the data set. For example, submit `nulls 3=y;` if you want the column that is associated with the third SAS variable to accept NULL values.

If you omit the NULLS statement, the DBMS default action occurs. You can list as many variables as you want in one NULLS statement. If you have previously defined a column as NULLS=N, you can use the NULLS statement to redefine it to accept NULL values.

**See:** ERRLIMIT= on page 991

**Syntax**

`NULLS variable-identifier-1 = Y | N <…variable-identifier-n = Y | N>;`
Details

The NULLS statement specifies whether the DBMS columns that are associated with the listed input SAS variables allow NULL values. Specify Y to accept NULL values. Specify N to reject NULL values and to require data in that column.

If you specify N for a numeric column, no observations that contain missing values in the corresponding SAS variable are loaded into the table. A message is written to the SAS log, and the current error count increases by one for each observation that is not loaded.

If a character column contains blanks (the SAS missing value) and you have specified N for the DBMS column, blanks are inserted. If you specify Y, NULL values are inserted.

QUIT Statement

Terminates the procedure without further processing

Restriction: Valid in the DBLOAD procedure (control statement)

Syntax

QUIT;

RENAMEN Statement

Renames DBMS columns

Requirements: The column-name argument must be a valid DBMS column name.
If the column name includes lowercase characters, special characters, or national characters, you must enclose the column name in single or double quotation marks.
If no quotation marks are used, the DBMS column name is created in uppercase.
Use rename 3="employeename" syntax to preserve case.

Interactions: DELETE, LABEL, RESET
The RENAME statement overrides the LABEL statement for columns that are renamed.

Notes: This statement changes the names of the DBMS columns that are associated with the listed SAS variables.
If you omit the RENAME statement, all DBMS column names default to the corresponding SAS variable names unless you specify the LABEL statement.

Tips: The variable-identifier argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents where to place the variable in the data set. For example, submit rename 3=employeename; if you want to rename the column associated with the third SAS variable.
You can list as many variables as you want in one RENAME statement.COLUMN is an alias for the RENAME statement.

Example: You can use this statement to include variables that you have previously deleted, as shown in this example:

delete 3;
rename 3=empname;
The DELETE statement drops the third variable. The RENAME statement includes the third variable and assigns the name EMPNAME and the default column type to it.

**Syntax**

RENAME variable-identifier-1 = '<column-name-1' '<…variable-identifier-n = '<column-name-n'>;'

---

**RESET Statement**

Resets column names and data types to their default values

**Requirement:** You must use the RESET statement after the LABEL statement for the LABEL statement to take effect.

**Interaction:** DELETE, LABEL, RENAME, TYPE

**Tip:** The variable-identifier argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, submit `reset 3;` if you want to reset the column associated with the third SAS variable.

**Syntax**

RESET ALL | variable-identifier-1 '<…variable-identifier-n'>;

**Details**

The RESET statement resets columns that are associated with the listed SAS variables to default values for the DBMS column name, column data type, and ability to accept NULL values. If you specify ALL, all columns are reset to their default values, and any dropped columns are restored with their default values. Here are the default values.

- **column name**
  - defaults to the SAS variable name, or to the SAS variable label (if you have used the LABEL statement).

- **column type**
  - is generated from the SAS variable format.

- **nulls**
  - uses the DBMS default value.

---

**SQL Statement**

Submits a dynamic, nonquery, DBMS-specific SQL statement to the DBMS

**Restriction:** You cannot create a DBMS table and reference it in your DBMS-specific SQL statements within the same PROC DBLOAD step. The new table is not created until the RUN statement is processed.

**Requirements:** You must enter the keyword SQL before each DBMS-specific SQL statement that you submit.

You must use DBMS-specific SQL object names and syntax in the DBLOAD SQL statement.
Note: You can use the DBLOAD statement to submit these DBMS-specific SQL statements, despite whether you create and load a DBMS table.

Tips: The SQL-statement argument can be any valid dynamic DBMS-specific SQL statement except the SELECT statement. However, you can enter a SELECT statement as a substatement within another statement, such as in a CREATE VIEW statement.

To submit dynamic, non-query DBMS-specific SQL statements to the DBMS without creating a DBMS table, use the DBMS= option, any database connection statements, and the SQL statement.

Example: This example grants UPDATE privileges to user MARURI on the DB2 SasDemo.Orders table.

```
proc dbload dbms=db2;
  in sample;
  sql grant update on sasdemo.orders to maruri;
run;
```

Syntax

```
DBMS-specific-SQL-statement;
```

**TABLE= Statement**

specifies the name of the DBMS table to be created and loaded into a DBMS database

Restriction: do not use this statement when you are submitting dynamic DBMS-specific SQL statements to the DBMS without creating and loading a table.

Requirements: This statement is required when you create and load or append to a DBMS table. It must follow other database connection statements such as DATABASE= or USER=.

The table name must be a valid table name for the DBMS. (See the DBMS-specific reference in this document for the syntax for your DBMS.)

You must specify a table name that does not already exist. If a table by that name exists, an error message is written to the SAS log, and the table specified in this statement is not loaded.

If your table name contains lowercase characters, special characters, or national characters, it must be enclosed in quotation marks.

Syntax

```
TABLE= 'DBMS-specific-syntax';
```

**TYPE Statement**

Changes default DBMS data types in the new table

Requirement: The column-type argument must be a valid data type for the DBMS and must be enclosed in quotation marks.

Note: This statement changes the default DBMS column data types that are associated with the corresponding SAS variables.
Tips: The variable-identifier argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, submit

```plaintext
type 3='char(17)';
```

if you want to change the data type of the DBMS column associated with the third SAS variable.

If you omit the TYPE statement, the column data types are generated with default DBMS data types that are based on the SAS variable formats. You can change as many data types as you want in one TYPE statement. See the DBMS-specific reference in this document for a complete list of default conversion data types for the DBLOAD procedure for your DBMS.

Syntax

```plaintext
TYPE variable-identifier-1 = 'column-type-1'
<...variable-identifier-n = 'column-type-n'>;
```

WHERE Statement

Loads a subset of data observations into the new table

Requirement: The SAS-where-expression must be a valid SAS WHERE statement that uses SAS variable names, not DBMS column names, as defined in the input data set.

See: SAS System Options: Reference (syntax)

Example: This example loads only the observations in which the COUNTRY SAS variable has the value of BRAZIL:

```plaintext
where country='Brazil';
```

Syntax

```plaintext
WHERE SAS-where-expression;
```

Example: Append a Data Set to a DBMS Table

Features: PROC DBLOAD option

APPEND Statements

PROC DBLOAD
TABLE=
COMMIT=
ERRLIMIT=

Database connection statements

USER
password
PATH

By omitting the APPEND option from the DBLOAD statement, you can use the PROC DBLOAD SQL statements to create a DBMS table and append to it in the same PROC DBLOAD step.
The following example appends new employee data from the NewEmp SAS data set to the Employees DBMS table. The COMMIT statement causes a DBMS commit to be issued after every 100 rows are inserted. The ERRLIMIT statement causes processing to stop after 10 errors occur.

```sas
proc dbload dbms=oracle data=newemp append;
   user=myusr1; password=mypw1; path='mysrv1';
   table=employees; commit=100; errlimit=10; load;
run;
```
Here is the recommended reading list for this title.

- Base SAS Procedures Guide
- Base SAS Utilities: Reference
- Learning SAS by Example: A Programmer’s Guide
- The Little SAS Book: A Primer
- The Little SAS Book for Enterprise Guide
- PROC SQL: Beyond the Basics Using SAS
- “Processing Multilingual Data with the SAS® 9.2 Unicode Server” [SAS Institute technical paper]
- SAS Companion for your operating environment
- SAS Component Objects: Reference
- SAS Data Set Options: Reference
- SAS DS2 Language Reference
- SAS Encoding: Understanding the Details
- SAS Formats and Informats: Reference
- SAS Functions and CALL Routines: Reference
- SAS High-Performance Analytics Infrastructure: Installation and Configuration Guide
- SAS In-Database Products: Administrator's Guide
- SAS In-Database Products: User's Guide
- SAS Intelligence Platform: Security Administration Guide
- SAS Language Reference: Concepts
- SAS Macro Language: Reference
- SAS SQL Procedure User's Guide
- SAS Statements: Reference
- SAS System Options: Reference
- SAS/ACCESS Interface to PC Files: Reference
• *SAS/SHARE User’s Guide*

• SAS offers instructor-led training and self-paced e-learning courses to help you get started with SAS/ACCESS and related SAS products. For more information about available courses, see [support.sas.com/training](http://support.sas.com/training).

For a complete list of SAS publications, go to [sas.com/store/books](http://sas.com/store/books). If you have questions about which titles you need, please contact a SAS Representative:

SAS Books  
SAS Campus Drive  
Cary, NC 27513-2414  
Phone: 1-800-727-0025  
Fax: 1-919-677-4444  
Email: sasbook@sas.com  
Web address: sas.com/store/books
Glossary

access descriptor
a SAS/ACCESS file that describes data that is managed by SAS, by a database management system, or by a PC-based software application such as Microsoft Excel, Lotus 1-2-3, or dBASE. After creating an access descriptor, you can use it as the basis for creating one or more view descriptors. See also view descriptor.

appliance
See data warehouse appliance.

bulk load
to load large amounts of data into a database object, using methods that are specific to a particular DBMS. Bulk loading enables you to rapidly and efficiently add multiple rows of data to a table as a single unit.

client
an application that requests either resources or services from a server, possibly over a network.

column
a vertical component of a table. Each column has a unique name, contains data of a specific type, and has particular attributes. A column is analogous to a variable in SAS terminology.

column function
an operation that is performed for each value in the column that is named as an argument of the function. For example, AVG(SALARY) is a column function.

commit
the process that ends a transaction and that makes permanent any changes to the database that the user made during the transaction. See also rollback.

data entity (entity)
an Rdb or VMS item that can be defined and manipulated using VAX SQL. Rdb and VMS entities include databases, tables, views, columns, and indexes.

data set
See SAS data set.
DATA step view
A type of SAS data set that consists of a stored DATA step program. A DATA step view contains a definition of data that is stored elsewhere; the view does not contain the physical data. The view's input data can come from one or more sources, including external files and other SAS data sets. Because a DATA step view only reads (opens for input) other files, you cannot update the view's underlying data.

data type (type)
an attribute of every column in a table or database, indicating the type of data in the column and how much physical storage it occupies.

data value
A unit of character, numeric, or alphanumeric information that is stored as a single item in a data record.

data view
See SAS data view.

data warehouse appliance (appliance)
an integrated set of servers, storage, one or more operating systems, DBMS, and software that is specifically pre-installed and pre-optimized for data warehousing.

database
An organized collection of related data. A database usually contains named files, named objects, or other named entities such as tables, views, and indexes.

database management system (DBMS)
a software application that enables you to create and manipulate data that is stored in the form of databases. See also relational database management system.

DB2 catalog
A set of tables that DB2 maintains for its own use. These tables contain information about the other objects your DB2 system manages.

DBMS
See database management system.

engine (SAS engine)
a component of SAS software that reads from or writes to a file. Various engines enable SAS to access different types of file formats.

entity
See data entity.

explicit pass-through
A form of the SQL pass-through facility that passes the user-written, DBMS-specific SQL query code directly to a particular DBMS for processing. See also implicit pass-through.

foreign key
A column or combination of columns in one table that references the corresponding primary key in another table. A foreign key must have the same attributes as the primary key that it references.

function
See SAS function.
Hadoop Distributed File System (HDFS)

A portable, scalable framework, written in Java, for managing large files as blocks of equal size. The files are replicated across multiple host machines in a Hadoop cluster in order to provide fault tolerance.

HDFS

See Hadoop Distributed File System.

Implicit pass-through

A form of the SQL pass-through facility that translates SAS SQL query code to the DBMS-specific SQL code, enabling the translated code to be passed to a particular DBMS for processing. See also explicit pass-through.

Index

See SAS index.

Informat

See SAS informat.

Interactive line mode (line mode)

A method of running SAS programs in which you enter one line of a SAS program at a time at the SAS session prompt. SAS processes each line immediately after you press the ENTER or RETURN key. Procedure output and informative messages are returned directly to your display device.

Interface view engine

A type of SAS engine that SAS/ACCESS software uses to retrieve data from files that have been formatted by another vendor's software. Each SAS/ACCESS interface has its own interface view engine, which reads the interface product data and returns the data in a form that SAS can understand (that is, in a SAS data set). See also engine.

Library member

Any of several types of SAS file in a SAS library. A library member can be a data set, a view, a catalog, a stored program, or an access descriptor.

Library reference

See libref.

Libref (library reference)

A SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library. See also SAS library.

Line mode

See interactive line mode.

Member name

A name that is assigned to a SAS file in a SAS library. See also member type.

Member type

A SAS name that identifies the type of information that is stored in a SAS file. Member types include ACCESS, AUDIT, DMBD, DATA, CATALOG, FDB, INDEX, ITEMSTOR, MDDB, PROGRAM, UTILITY, and VIEW.
missing value
a type of value for a variable that contains no data for a particular row or column. By
default, SAS writes a missing numeric value as a single period and a missing
character value as a blank space. See also null value.

null value
a special value that indicates the absence of information. Null values are analogous
to SAS missing values. See also missing value.

object
any entity that can be manipulated by the commands of a programming language.
Examples are values, variables, functions, and data structures.

observation
a row in a SAS data set. All of the data values in an observation are associated with a
single entity such as a customer or a state. Each observation contains either one data
value or a missing-value indicator for each variable.

pass-through facility
See SQL pass-through facility.

predicate
a component of a SAS WHERE statement or a component of an SQL WHERE or
HAVING clause that is used in view descriptor selection criteria.

PROC SQL view
a SAS data set that is created by the SQL procedure. A PROC SQL view contains no
data. Instead, it stores information that enables it to read data values from other files,
which can include SAS data files, SAS/ACCESS views, DATA step views, or other
PROC SQL views. The output of a PROC SQL view can be either a subset or a
superset of one or more files. See also SAS data view.

query
a set of instructions that requests particular information from one or more data
sources.

RDBMS
See relational database management system.

referential integrity
a set of rules that a DBMS uses to ensure that whenever a data value in one table is
changed, the appropriate change is also made to any related values in other tables or
in the same table. Referential integrity is also used to ensure that related data is not
deleted or changed accidentally.

relational database management system (RDBMS)
a database management system that organizes and accesses data according to
relationships between data items. The main characteristic of a relational database
management system is the two-dimensional table. Examples of relational database
management systems are DB2, Oracle, Sybase, and Microsoft SQL Server.

rollback
a data recovery process that restores a database after a hardware or software failure,
or that returns it to a state before changes were made. See also commit.
SAS data file
a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data. See also SAS data set, SAS data view.

SAS data set (data set)
a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views.

SAS data view (data view)
a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns) plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats.

SAS engine
See engine.

SAS file
a specially structured file that is created, organized, and maintained by SAS. A SAS file can be a SAS data set, a catalog, a stored program, an access descriptor, a utility file, a multidimensional database file, a financial database file, a data mining database file, or an item store file.

SAS function (function)
a type of SAS language element that is used to process one or more arguments and then to return a result that can be used in either an assignment statement or an expression.

SAS index (index)
a component of a SAS data set that enables SAS to access observations in the SAS data set quickly and efficiently. The purpose of SAS indexes is to optimize WHERE-clause processing and to facilitate BY-group processing.

SAS informat (informat)
a type of SAS language element that is used to read data values according to the data's type: numeric, character, date, time, or timestamp.

SAS library
one or more files that are defined, recognized, and accessible by SAS, and that are referenced and stored as a unit. Each file is a member of the library.

SAS metadata
metadata that is created by SAS software. Metadata that is in SAS Open Metadata Architecture format is one example.

SAS variable (variable)
a column in a SAS data set or in a SAS data view. The data values for each variable describe a single characteristic for all observations (rows).

scalar function
in DB2 and SQL/DS, an SQL function that is applied to each value in the column named as the argument of the function (for example, MONTH(BILLEDON)=11).
serde
an interface that enables serialization or deserialization of one or more file formats.

server
software that provides either resources or services to requesting clients, possibly over a network.

SQL
See Structured Query Language.

SQL pass-through facility (pass-through facility)
the technology that enables SQL query code to be passed to a particular DBMS for processing.

SSID
See subsystem ID.

Structured Query Language (SQL)
a standardized, high-level query language that is used in relational database management systems to create and manipulate objects in a database management system. SAS implements SQL through the SQL procedure.

subsystem ID (SSID)
the unique name that identifies each DB2 subsystem on your machine.

table space
a named DB2 object that identifies sets of pages where records for one or more tables are stored. (A page is a 4K-byte or 32K-byte unit of storage within a table space.)

temporal data
event data that occurs at a particular date and time, such as an account inquiry. Temporal data is often referred to as time-sensitive data.

trigger
a type of user-defined stored procedure that is executed whenever a user issues a data-modification command such as INSERT, DELETE, or UPDATE for a specified table or column. Triggers can be used to implement referential integrity or to maintain business constraints. See also referential integrity.

type
See data type.

variable
See SAS variable.

view
da definition of a virtual data set that is named and stored for later use. A view contains no data; it merely describes or defines data that is stored elsewhere.

view descriptor
a SAS/ACCESS file that defines part or all of the DBMS data that is described by an access descriptor. See also access descriptor.
windowing procedure

a SAS procedure that you can use by entering information in one or more windows or dialog boxes. For example, the FSVIEW procedure is a windowing procedure. Some procedures, such as ACCESS and DBLOAD, can be used either as windowing procedures or in batch mode.
Index

Special Characters
_ALL_ argument 97
LIBNAME statement 97
Sybase 871
%SUPERQ macro 446

A
ACCDESC=
   DBLOAD procedure 987
   PROC ACCESS statement 959
access descriptors 970
   ACCESS procedure with 970
      creating 68, 961, 987
data set and descriptor access 971
   identifying DBMS table for 968
   listing columns in, with information
      name, for converting 977
      resetting columns to default settings
         966
      updating 969, 972
   access levels
      for opening libref connections 102
   access methods 4
      selecting 4
ACCESS procedure
   accessing DBMS data 955
   database connection statements 959
   DBMS specifics 957
   descriptors with 970
   examples 972
   relational databases 955
   syntax 957
ACCESS procedure, relational databases
   4
   DB2 under z/OS 544
   DB2 under z/OS data conversions 571
   how it works 68
   names 15
   Oracle 792
   Oracle data conversions 809
   overview 68
   reading data 68
   Sybase 858
   Sybase data conversions 869
   unsupported in Teradata 892
   ACCESS= LIBNAME option 102
   accessing data
      from DBMS objects 66
      repeatedly accessing 39
ACCOUNT= connection option
   Teradata 894
   acquisition error tables 369, 370
   additional options
      for IMPORT statement 290
      SFTP access 312
ADJUST_BYTE_SEMANTIC_COLUMN_LENGTH=
   S= LIBNAME option 103
ADJUST_NCHAR_COLUMN_LENGTH=
   S= LIBNAME option 102
aggregate functions
   passing to DBMS 44
ALL option
   LIST statement (ACCESS) 964
   RESET statement (ACCESS) 966
ALLOWED_SQLCODES= LIBNAME option
   105
ames
   case sensitivity 14
   ANSI outer-join syntax 217
   appending data sets 997
   application identifier
      for DB2 monitoring 194
   applications
      threaded 54, 60
   ARRAY data type
      Hive (Hadoop) 616
AS/ACCESS
   features for common tasks 5
ASSIGN
   ACCESS procedure 957
   ASSIGN statement
      ACCESS procedure 960
Aster 486
autopartitioning scheme 494
bulk loading 497
data conversions 500
data set options 490
data types 499
DBSLICE= data set option 495
DBSLICEPARM= LIBNAME option 495
LIBNAME statement 486
nullable columns 494
numeric data 499
passing joins to 496
passing SAS functions to 495
special catalog queries 493
SQL pass-through facility 492
string data 499
WHERE clauses 494
attachment facility 559, 580
known issues with 578
AUTHDOMAIN= LIBNAME option 105
authentication domain metadata objects
connecting to server with name of 105
AUTHID= data set option 242
AUTHID= LIBNAME option 106
authorization
client-server, DB2 under z/OS 575
authorization ID 242
qualifying table names with 106
autocommit 242
MySQL 703
AUTOCOMMIT= data set option 242
AUTOCOMMIT= LIBNAME option 107
automatic COMMIT 327
after specified number of rows 132
autopartitioning 59
Aster 494
column selection for MOD partitioning 541
configuring DB2 EEE nodes on physically partitioned databases 515
DB2 under UNIX and PC Hosts 513
DB2 under z/OS 540
Greenplum 592
HAWQ 645
Informix 671
ODBC 743
Oracle 789
restricted by WHERE clauses 541
SAP HANA 838
Sybase 856
Sybase IQ 881
Teradata 914

B
B2 under UNIX and PC Hosts
configuring EEE nodes on physically partitioned databases 515
backslash
in literals 161, 357
backslash character
literal or escape character 134
batch
records to commit 290
BCP_OPTIONS
BL_IMPORT_OPTIONS= data set option 290
BCP=
BULKLOAD= data set option 317
big integer
integers, big 234
BIGINT data type
DB2 under UNIX and PC Hosts 527
Greenplum 600
HAWQ 645
Impala 659
Netezza 724
Sybase IQ 888
BIGINT data types
Hadoop 616
BIGINT_WARN=
WARN_BIGINT= LIBNAME option 234
binary data
DB2 under UNIX and PC Hosts 322
Oracle 803
Sybase 871
BINARY data type
Hive (Hadoop) 616
Binary data types
DB2 under UNIX and PC Hosts 527
Greenplum 600
binary string data
SAP HANA 842
Teradata 931
Vertica 951
BINARY_DOUBLE data type
Oracle 802
BINARY_FLOAT data type
Oracle 802
BIT data type
Sybase 866
Sybase IQ 888
BL_ALLOW_READ_ACCESS= data set option 243
BL_ALLOW_WRITE_ACCESS= data set option 243
BL_API_BULKLOAD= data set option 244
BL_BADFILE
BL_CONTROL_FIELD_DELIMITER= data set option 247
BL_FILE_BADFILE= data set option 284
BL_FILE_CONTROLFILE= data set option 285
BL_FILE_DATAFILE= data set option 286
BL_FILE_DEFAULT_DIR= data set option 286
BL_FILE_DELETE_DATAFILE= data set option 287
BL_FILE_LOG
BL_FILE_BADFILE= data set option 284
BL_FORCE_NOT_NULL= data set option 287
BL_FORMAT= data set option 288
BL_HEADER= data set option 289
BL_HOST= data set option 289
BL_HOST= LIBNAME option 108
BL_IMPORT_BATCH_SIZE= data set option 290
BL_IMPORT_OPTIONS= data set option 290
BL_FIELD_CONTROLFILE= data set option
BL_INDEX_OPTIONS= data set option 292
BL_INDEXING_MODE= data set option 293
BL_KEEPIDENTITY= data set option 294
BL_KEEPIDENTITY= LIBNAME option 109
BL_KEEPNULLS= data set option 294
BL_KEEPNULLS= LIBNAME option 109
BL_LOAD_METHOD= data set option 295
BL_LOAD_REPLACE= data set option 296
BL_LOCATION= data set option 296
BL_LOG
BL_FILE_BADFILE= data set option 284
BL_LOG= data set option 296, 910
BL_LOG= LIBNAME option 110
BL_MAPPFILE= data set option 297
BL_METHOD= data set option 298
BL_MODIFIED_BY
BL_IMPORT_OPTIONS= data set option 290
BL_NULL= data set option 299, 300
BL_OPTIONS
BL_IMPORT_OPTIONS= data set option 290
BL_OPTIONS= data set option 300
BL_OPTIONS= LIBNAME option 110
BL_PARFILE= data set option 301
BL_PATH= data set option 302
BL_PORT_MAX= data set option 304
BL_PORT_MIN= data set option 304
BL_PORT= data set option 303
BL_PORT= LIBNAME option 111
BL_PRESERVE_BLANKS= data set option 305
BL_PROTOCOL= data set option 305
BL_PSQL_PATH= data set option 306
BL_QUOTATION
BL_CONTROL_QUOTATION_MARK = data set option 248
BL_QUOTE
BL_CONTROL_QUOTATION_MARK = data set option 248
BL_QUOTE= data set option 306
BL_RECORD_DELIMITER
BL_CONTROL_RECORD_DELIMITER R= data set option 248
BL_RECOVERABLE= data set option 307
BL_REJECT_LIMIT= data set option 308
BL_REJECT_TYPE= data set option 308
BL_REMOTE_FILE= data set option 309
BL_RETURN_WARNINGS_AS_ERROR RS= data set option 310
BL_SERVER
BL_SFT_HOST= data set option 311
BL_SERVER_DATAFILE= data set option 310
BL_SFTP_HOST= data set option 311
BL_SFTP_OPTIONS= data set option 312
BL_SFTP_SERVER
BL_SFT_HOST= data set option 311
BL_SFTP_USER= data set option 312
BL_SFTP_WAIT_MILLISECONDS= data set option 313
BL_SQLLDR_PATH= data set option 313
BL_SUPPRESS_NULLIF= data set option 314
BL_TABLE_LOCK
BL_IMPORT_TABLE_LOCK= data set option 291
BL_TYPE_CHECK
BL_IMPORT_TYPE_CHECK= data set option 291
BL_USE_PIPE= data set option 315
PostgreSQL 820
protocol for 305
quotation character for CSV mode 248, 306
rejected records 245, 284
row warnings 316
SAP HANA 841
saving copy of loaded data 251
segment instances 282
skip or load first record in input data file 289
SQL*Loader Index options 292
stopping gpfdist 596, 642
string that replaces a null value 300
Sybase 862
Sybase IQ 884
SYSDISC data set name for LOAD utility 259
SYSERR data set name for LOAD utility 259
SYSSN data set name for LOAD utility 259
troubleshooting gpfdist 597, 643
unit address for permanent data sets 258
visibility of original table data 243
warnings 310
bulk loading partitioned tables 391
bulk rows buffering for output 317
bulk unloading 112
Netezza 722
unloading rows of data as one unit 318
bulk-load facility
DBMS-specific 112
deleting the data file 272, 287
passing options to 110
BULK_BUFFER= data set option 317
BULKCOPY= database description statement
Sybase 860
BULKCOPY= statement, DBLOAD procedure
ODBC 747
bulkload
additional options 290
check data types 291
directory for temporary files 286
table from an existing data set 253
BULKLOAD_HOST=
BL_HOST_LIBNAME option 108
BULKLOAD_PORT=
BL_PORT=data set option 303
BL_PORT=LIBNAME option 111
BULKLOAD=
FASTLOAD= LIBNAME option 163
BULKLOAD= data set option 317
BULKLOAD= LIBNAME option 112
bulkloading
directory for temporary files 286
BULKUNLOAD= data set option 318
BULKUNLOAD= LIBNAME option 112
BUSINESS_DATATYPE= data set option 319
BUSINESS_TIMEFRAME= data set option 319
BY clause
ordering query results 919
replacing SORT procedure with Teradata 919, 921
BYTE data type
Teradata 931
byte semantics
specifying CHAR or VARCHAR data type columns 102
BYTEA data type
Greenplum 600
BYTEINT data type 114
Netezza 724
Teradata 931
bytes
for creating an Oracle table 128
maximum for database server encoding 150

C escape sequences 280
CAF (Call Attachment Facility) 580
case sensitivity
DBMS column names 397
FastExport Utility 915
for user name 193
MySQL 707
names 14
Sybase 865
CAST_OVERHEAD_MAXPERCENT= data set option 321
CASTOVERHEAD_MAXPERCENT= LIBNAME option 115
CAST= data set option 320
CAST= LIBNAME option 113
casting
overhead limit for 115
performed by SAS or Teradata 113
catalog queries
DB2 under UNIX and PC Hosts 512
Greenplum 591
HAWQ 637
Netezza 718
ODBC 741
OLE DB 765
Sybase IQ 880
catalog tables
  overriding default owner of 448
CCESS procedure, relational databases
  updating data 69
CELLPROP= LIBNAME option 115
CFG= connection option
  Hadoop 607
CHAR column lengths 128
CHAR data type
  DB2 under UNIX and PC Hosts 527
  DB2 under z/OS 568
  Greenplum 599
  HAWQ 645
  Impala 659
  Netezza 724
  Oracle 802
  Sybase 866
  Sybase IQ 888
  Teradata 931
CHAR data type columns
  adjusting default for 116
  adjusting lengths for 102
  specified with byte semantics 102
CHAR_AS_BINARY= data set option 322
CHAR_AS_NCHAR= LIBNAME option 116
CHAR(n) data type
  Greenplum 599
  HAWQ 645
  Hive (Hadoop) 616
character data
  codepage for converting during bulk loading 246
  DB2 under UNIX and PC Hosts 527
  Informix 674
  length of 341
  MySQL 707
  Netezza 724
  Oracle 802
  Sybase 866
character data type
  length of very long types 142
character set encoding
  for bulk loading an external table 279
character set metadata
  transfer from input to output for processing 207
character string data
  Teradata 931
character to use for escape sequence 280
characters
  replacing unsupported characters in names 19
CHECKPOINT=
  DBCOMMIT= data set option 327
  DBCOMMIT= LIBNAME option 132
  CHECKPOINT= LIBNAME option 97
  FastLoad and 910
  checkpoints 419
  interval between 368
  restart table 372
CLEAR argument
  LIBNAME statement 97
client encoding
  column length for 131
  fixed-width column length for 129
client view
  for bulk-loading data files 246
client-server authorization
  DB2 under z/OS 575
  client/server authorization
  known issues with RRSAF support 578
  non-libref connections 577
CLOB data type
  DB2 under UNIX and PC Hosts 527
  DB2 under z/OS 569
  Oracle 802
codepages 246
codes, SQL
  to ignore (DB2) 105
COL_DELIM=
  COLUMN_DELIMITER= data set option 322
COL_DELIMITER=
  COLUMN_DELIMITER= data set option 322
column data
  CHAR or VARCHAR data type
  columns 116
column labels
  returned by engine 345
  specifying for engine use 148
column length
  CHAR or VARCHAR columns 102
  fixed-width for client encoding 129
  fixed-width for database server encoding 148
  for CHAR 128
  for client encoding 131
  for VARCHAR2 128
  NCHAR or NVARCHAR columns 103
column name as partition key 390
column names
  as partition key for creating fact tables 183, 391
  basing variable names on 969
defaulting to labels 987
embedded spaces and special characters 199
in DISTRIBUTE ON clause 354
naming during output 338
preserving 20, 185, 397
renaming 16, 138, 475, 987
COLUMN_DELIMITER= data set option 322
columns
changing column formats 962
CSV column values 287
date format of 404
distributing rows across database
segments 355
ignoring read-only columns 165
limiting retrieval 37
NULL as valid value 342
NULL values accepted in 987
NULL values and bulk loading 109
renaming 334
resetting to default settings 966
selecting 967
selecting for MOD partitioning 541
command restrictions for SQL
for Informix stored procedures 669
COMMAND_TIMEOUT= data set option 323
COMMAND_TIMEOUT= LIBNAME option 117
commands
timeout for 117, 323
commit
records to insert 290
COMMIT, automatic 327
after specified number of rows 132
COMMIT= DBLOAD procedure 987
COMMIT= option
PROC DB2UTIL statement 551
committed reads
Informix 673
COMPLETE= connection option
DB2 under UNIX and PC Hosts 505
Microsoft SQL Server 683
ODBC 733
OLE DB 758
complex data
Hive (Hadoop) 616
CONFIG=
CFG= connection option for Hadoop 607
CONFIGDIR= data set option 323, 324
CONFIGDIR= LIBNAME option 117, 118
configuration 117, 118
DB2 EEE nodes on physically partitioned databases 515
file server 596, 642
SQL Server partitioned views for
DBSLICE= 744
configuration directory 323, 324
configuration options
SFTP access 312
connect exits 33
CONNECT statement
SQL procedure 473
CONNECT statement, SQL procedure arguments 473
example 477
connection groups 124
connection information
prompts to enter 146
protecting 30
connection options
for NLS 63
Impala 654
LIBNAME statement 97
Netezza 718
CONNECTION TO component 477
syntax 472
CONNECTION_GROUP= argument
CONNECT statement 474
CONNECTION_GROUP= LIBNAME option 124
CONNECT= argument
CONNECT statement 474
CONNECTION= LIBNAME option 118
connections
CONNECT statement for establishing 473
controlling DBMS connections 31
DB2 under z/OS, optimizing 556
librefs and 118
simultaneous, maximum number allowed 175
specifying when connection occurs 153
terminating 480
to DBMS server for threaded reads 326
utility connections 234
with name of authentication domain
metadata object 105
control statements
file for 249, 255, 285
conversion of data
from Hive (Hadoop) to SAS 618
from SAS to Hive (Hadoop) 619
conversions
Impala to SAS 660
SAS to Impala 661
converting descriptors to SQL views 975
CORRELATION_ID
PROGRAM_NAME= LIBNAME option 194
CREATE
ACCESS procedure 957
CREATE statement
ACCESS procedure 961
SQL procedure 481
CREATE TABLE statement 332
adding DBMS-specific syntax to 137
creating and deleting Hive tables 136, 330
Cross Memory Services (XMS) 578
CSV column values 287
CSV mode
quotation character for 248, 306
currency control 32
Cursor Library 233
cursor stability reads
Informix 673
cursor type 125, 324
CURSOR_TYPE= data set option 324
CURSOR_TYPE= LIBNAME option 125, 233
CV2VIEW procedure 975
converting a library of view descriptors 983
converting an individual view descriptor 980
examples 980
syntax 976
CV2VIEW Procedure
relational databases 975
cylinders
LOAD utility 267

D
data
read with JDBC 203, 402
user-defined (Sybase) 867
data access
from DBMS objects 66
repeatedly accessing 39
data buffers
MultiLoad 374
transferring data to Teradata 316, 368
data class for SMS-managed data set 258
data classes
for SMS-managed data sets 258
data conversion
from Hive (Hadoop) to SAS 618
Hive to SAS (Hadoop) 617
SAS to Hive (Hadoop) 619
data conversion issues
Impala to SAS 660
data conversions
Aster 500
DB2 under UNIX and PC Hosts,
DBLOAD procedure 529
DB2 under UNIX and PC Hosts,
LIBNAME statement 528
DB2 under z/OS, ACCESS procedure 571
DB2 under z/OS, DBLOAD procedure 573
DB2 under z/OS, LIBNAME statement 570
Greenplum 601
HAWQ 646
Impala to SAS 660
Informix, LIBNAME statement 676
Informix, SQL pass-through facility 677
Microsoft SQL Server 693
MySQL 708
Netezza 725
ODBC 753
OLE DB 776
Oracle, ACCESS procedure 809
Oracle, DBLOAD procedure 810
Oracle, LIBNAME statement 807
overhead limit 115
overhead limit for performing in Teradata instead of SAS 321
performed by SAS or Teradata 113
PostgreSQL 823
SAP HANA 846
SAS to Impala 661
Sybase IQ 889
Sybase, ACCESS procedure 869
Sybase, DBLOAD procedure 870
Sybase, LIBNAME statement 867
Teradata 113, 320, 321, 932
Teradata DBMS server versus SAS 320
data conversions for Impala troubleshooting 660
data copy
preserving backslashes in literals 161
data extraction
numeric precision and 9
data functions 67
data ordering
threaded Reads and 60
data providers
connecting directly to (OLE DB) 761
data representation
numeric precision and 8
data security 28
See also security
controlling DBMS connections 31
customizing connect and disconnect exits 33
Index

data source commands
  timing out 117
data sources
  default login timeout 174
  schemas 406
  updating and deleting rows in 231
DATA step views 6
data transfer
  named pipes for 315
data types
  Aster 499
  changing default 987
  checking before bulkload 291
  DB2 under UNIX and PC Hosts 527
  DB2 under z/OS 568
  DBMS columns 987
  for NLS 64
  Greenplum 599
  Hadoop 615
  HAWQ 645
  Hive (Hadoop) 616
  Impala 659
  Informix 674
  Microsoft SQL Server 693
  MySQL 707
  Netezza 723
  ODBC 752
  OLE DB 775
  Oracle 802
  overriding SAS defaults 346
  PostgreSQL 823
  resetting to default 987
  SAP HANA 842
  SAS 659
  specifying 351
  Sybase 886
  Sybase IQ 887
  temporal, Teradata 934
  Teradata 931
  transfer from input to output for processing 207
  Vertica 950, 951
data types, default
  for numeric table columns 178
DATA= option
  PROC DB2UTIL statement 550
database administrators
  DB2 under z/OS 578
  privileges and 27
Database Connection
  ACCESS procedure 957
database connection statements
  ACCESS procedure 959
  DBLOAD procedure 990
database links 339
database name
for bulk loading 271
database objects 129
identifying with qualifiers 196
linking from local database to database objects on another server 142
qualifiers when reading 398
database server encoding
fixed-width column length for 148
maximum bytes for 150
database servers
Informix 677
librefs pointing to 205
DATABASE=
DATASRC= connection option for DB2 505
DATASRC= connection option for Microsoft SQL Server 682
DATASRC= connection option for ODBC 732
SCHEMA= connection option for Impala 651
SCHEMA= data set option 406
DATABASE= connection option
Aster 486
Greenplum 584
HAWQ 630
MySQL 698
Netezza 712
PostgreSQL 812
Sybase 851
Sybase IQ 875
Teradata 895
Vertica 940
DATABASE= database description statement
Sybase 860
Databases
linking from default to another database on a connected server 142
linking from local database to database objects on another server 142
DATASETS procedure
assigning passwords 29
reading Oracle table names 193
showing synonyms 211, 219
specify database objects 129
DATASOURCE= connection option
OLE DB 756
DATASRC= connection option
DB2 under UNIX and PC Hosts 505
Microsoft SQL Server 682
ODBC 732
date and time data types
Impala 659
DATE data type
casting 114
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 569
Greenplum 600
HAWQ 646
Netezza 724
Oracle 803
Sybase 866
Sybase IQ 888
Teradata 931
date formats
of DBMS columns 404
date values
Hive (Hadoop) 616
date, time, and interval data
Informix 675
date, time, and money data
Sybase 866
date, time, and timestamp data
SAP HANA 842
Sybase IQ 888
Teradata 931
date, time, and timestamp values
Netezza 724
DATETIME data type
Sybase 866
datetime values
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 569
Greenplum 600
HAWQ 646
reading as character strings or numeric date values 218
Vertica 951
DATETIME2= data set option 326
DATETIME2= LIBNAME option 127
DB_LENGTH_SEMANTICS_BYTE=
LIBNAME option 128
DB_OBJECTS= LIBNAME option 129
DB_ONE_CONNECT_PER_THREAD=
data set option 326
DB=
DATABASE= connection option for
Aster 486
DATABASE= connection option for Greenplum 584
DATABASE= connection option for HAWQ 630
DATABASE= connection option for Netezza 712
DATABASE= connection option for PostgreSQL 812
DATABASE= connection option for Sybase 851
DATABASE= connection option for Sybase IQ 875
DATABASE= connection option for Teradata 895
DATABASE= connection option for Vertica 940
DATABASE= database description statement for Sybase 860
DATASRC= connection option for DB2 505
DATASRC= connection option for Microsoft SQL Server 682
DATASRC= connection option for ODBC 732
SCHEMA= connection option for Impala 651

DB2
appending versus replacing rows during bulk loading 296
bulk-loading data file as seen by server instance 310
overriding owner of catalog tables 448
parallelism 154
saving copy of loaded data 251
server data file 310
DB2 catalog tables
overriding default owner of 448
DB2 monitoring 194
DB2 SELECT statement 257
DB2 subsystem identifier 560
DB2 subsystem name 560
DB2 tables 559
database and table space for 559
deleting rows 549, 553
inserting data 553
inserting rows 549
modifying data 552
updating rows 549
DB2 under UNIX and PC Hosts 504, 527
autopartitioning scheme 513
bulk loading 521
calling stored procedures 516
capturing bulk-load statistics into macro variables 523
data conversions, DBLOAD procedure 529
data conversions, LIBNAME statement 528
data set options 508
data types 527
date, time, and timestamp data 527
DBLOAD procedure 518
DBSLICE= data set option 514
DBSLICEPARM= LIBNAME option 514
LIBNAME statement 504
locking 525
maximizing load performance 523
naming conventions 526
NULL values 528
nullable columns 513
numeric data 527
passing joins to 521
passing SAS functions to 520
special catalog queries 512
SQL pass-through facility 511
string data 527
supported features 78
WHERE clauses 513
DB2 under z/OS 533
ACCESS procedure 544
accessing system catalogs 581
attachment facilities 580
autopartitioning scheme 540
bulk loading 561
bulk loading, file allocation and naming 562
calling stored procedures 542
client-server authorization 575
column selection for MOD partitioning 541
creating SAS data sets from 547
data conversions, ACCESS procedure 571
data conversions, DBLOAD procedure 573
data conversions, LIBNAME statement 570
data set options 536
data set options for bulk loading 562
data types 568
database administrator information 578
DB2 subsystem identifier 560
DB2EXT procedure 547
DB2UTIL procedure 549
DBLOAD procedure 546
DBSLICE= data set option 542
DBSLICEPARM= LIBNAME option 542
DDF Communication Database 579
deleting rows 549
how it works 578
inserting rows 549
Large object (LOB) data 569
LIBNAME statement 533
locking 566
modifying DB2 data 552
naming conventions 567
NULL values 569
numeric data 568
optimizing connections 556
passing joins to 558
passing SAS functions to 557
performance 554
database connection statements 990
DBMS specifics 987
example 997
properties of 986
relational databases 985
sending data from SAS to DBMS 985
syntax 987
DBLOAD procedure, relational databases 4
DB2 under UNIX and PC Hosts 518
DB2 under UNIX and PC Hosts data conversions 529
DB2 under z/OS 546
DB2 under z/OS data conversions 573
how it works 69
Microsoft SQL Server 690
names 16
ODBC 747
Oracle 794
Oracle data conversions 810
Sybase 859
Sybase data conversions 870
unsupported in Teradata 892
DBMASTER= data set option 340
DBMAX_TEXT= argument 475
CONNECT statement 475
DBMAX_TEXT= data set option 341
DBMAX_TEXT= LIBNAME option 142
DBMS
assigning libref to remote DBMS 101
autocommit capability 242
connecting to, with CONNECT statement 473
failures when passing joins to 47
LIBNAME statement specifics 96
passing DISTINCT and UNION processing to 48
passing functions to, with SQL procedure 44
passing functions to, with WHERE clauses 49
passing joins to 45
passing WHERE clauses to 49
pushing heterogeneous joins to 41
submitting SQL statements to 987
DBMS connections
sharing 121
DBMS data
creating DBMS tables 24
extracting to data sets 30
pushing updates 42
renaming 16
repeatedly accessing 17
retrieving 17
retrieving and using in queries or views 477
SAS views of 6
sorting 39
subsetting and ordering 328
DBMS engine
code page for converting character data 246
trace information generated from 455
DBMS objects
accessing data from 66
naming behavior when creating 18
DBMS security 27
privileges 27
triggers 28
DBMS server
interrupting SQL processes on 179
number of connections for threaded reads 326
DBMS tables 99
See also tables
access descriptors based on 987
access methods 4
appending data sets to 997
committing or saving after inserts 987
creating 69
creating and loading 987
dropping variables before creating 987
inserting data with DBMS facility 112
limiting observations loaded to 987
loading data subsets into 987
locking data 229
naming 987
preserving column names 397
preserving guid 186
preserving names 191
reading from 99
verifying indexes 335
writing to 99
DBMS-specific
locking 33
DBMS= option
PROC ACCESS statement 959
DBMSTEMP= LIBNAME option 143
DBNULL= data set option 342
DBNULLKEYS= data set option 343
DBNULLKEYS= LIBNAME option 145
DBPROMPT= argument
CONNECT statement 475
DBPROMPT= data set option 344
DBPROMPT= LIBNAME option 146
DBSASLABEL= data set option 345
DBSASLABEL= LIBNAME option 148
DBSASTYPE= data set option 346
DBSERVER_ENCODING_FIXED= LIBNAME option 148
DBSERVER_MAX_BYTES= LIBNAME option 150
DBSLICE= data set option 347
Aster 495
configuring SQL Server partitioned views for 744
DB2 under UNIX and PC Hosts 514
DB2 under z/OS 542
Greenplum 592
HAWQ 638
Informix 672
ODBC 744
SAP HANA 838
Sybase 347
Sybase IQ 347, 882
threaded Reads and 55
DBSLICEPARM= data set option 349
threaded Reads and 55
DBSLICEPARM= LIBNAME option 151
Aster 495
DB2 under UNIX and PC Hosts 514
DB2 under z/OS 542
Greenplum 592
HAWQ 638
Informix 671
ODBC 744
SAP HANA 838
Sybase IQ 882
DBSLICEPARM= system option 451
DBSRVTP= system option 453
DBTERMCMD
DBCONTERM= LIBNAME option 135
DBTYPE= data set option 351
DDF Communication Database 579
debugging
tracing information for 227
DECIMAL data type
casting 114
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 568
Greenplum 600
HAWQ 645
Hive (Hadoop) 616
Netezza 724
Sybase 866
Sybase IQ 888
Teradata 931
DECPPOINT= option 559
default database
linking to another database on a connected server 142
default login timeout 174
DEFE R= argument
CONNECT statement 476
DEFE R= LIBNAME option 153
degree of numeric precision 7
DEGREE= data set option 353
DEGREE= LIBNAME option 154
DELETE
DBLOAD procedure 987
delete rules
MySQL 703
SAP HANA 837
Vertica 947
DELETE statement
SQL procedure 156, 482
SQL procedure, passing to empty a table 46
DELETE_MULT_ROWS= LIBNAME option 155
deleting and creating Hive tables 136
DELIM
BL_DELIMITER= data set option 275
DELIMITER
BL_DELIMITER= data set option 275
delimiters 322, 404
bulk loading 247, 248, 275
delimiting identifiers 199
descriptor files
ACCESS procedure with 970
creating 961
listing columns in, with information 964
resetting columns to default settings 966
updating 969
descriptors 970
ACCESS procedure with 970
dimension tables 155, 354
DIMENSION= data set option 354
DIMENSION= LIBNAME option 155
DIRECT option
SQL*Loader 277
DIRECT_EXE= LIBNAME option 156
DIRECT_SQL= LIBNAME option 47, 157
directory, default
for bulkloading 286
dirty reads
DB2 under UNIX and PC Hosts 525
Informix 673
Microsoft SQL Server 692
ODBC 751
OLE DB 772
PostgreSQL 822
Sybase IQ 886
Vertica 949
disconnect exits 33
DISCONNECT statement
SQL procedure 480
DISTINCT operator
passing to DBMS 48
DISTRIBUTE ON clause
column names in 354
DISTRIBUTE_ON= data set option 354
DISTRIBUTE=
    DISTRIBUTE_ON= data set option 354
Distributed Relational Database
    Architecture (DRDA) 580
DISTRIBUTED_BY= data set option 355
DOUBLE data type
    DB2 under UNIX and PC Hosts 527
    DB2 under z/OS 568
    Hive (Hadoop) 616
    Impala 659
    Netezza 724
    Sybase IQ 888
DOUBLE PRECISION data type
    DB2 under UNIX and PC Hosts 527
    DB2 under z/OS 568
    Greenplum 600
    HAWQ 645
    Netezza 724
    Sybase IQ 888
double quotation marks
    naming and 17
    DQUOTE=ANSI option
    naming behavior and 17, 22
    DRDA (Distributed Relational Database
    Architecture) 580
Driver Manager
    ODBC Cursor Library and 233
driver specification 233
    Impala 233
DRIVER_VENDOR= LIBNAME option 159
DRIVER= connection option
    SAP HANA 830
DROP
    ACCESS procedure 957
    DROP statement
        ACCESS procedure 962
        SQL procedure 482
DROP= data set option
    limiting retrieval 38
DSN=
    DATASRC= connection option for DB2 505
    DATASRC= connection option for
        Microsoft SQL Server 682
    DATASRC= connection option for
        ODBC 732
DSN= connection option
    Aster 487
    Greenplum 585
    HAWQ 631
    Netezza 713
    PostgreSQL 813
SAP HANA 830
    Sybase IQ 875
    Vertica 941
    DSN= statement, DBLOAD procedure
    Microsoft SQL Server 690
    ODBC 747
duplicate rows 409
dynamic data
    accessing in web tables 597, 643
E
    EEE nodes
        configuring on physically partitioned
        databases 515
    embedded LIBNAME statements
        SQL views with 99
    ENABLE_BULK= LIBNAME option 159
    encoding
        character set encoding for bulk loading
        an external table 279
column length for client encoding 131
    fixed-width column length for client
    encoding 129
    fixed-width column length for database
    server encoding 148
    maximum bytes for database server
    encoding 150
    maximum bytes per single character in
    server encoding 150
    ENCRIPT= connection option
        SAP HANA 831
        encryption 421
        Teradata 922
    engines
        blocking operations and 453
column labels used by 148
generating trace information from
    DBMS engine 455
    environment variable, Boolean
        OLE DB 776
    environment variables
        TRUNCATE_BIGINT 724
    ERRLIMIT statement
        DB2UTIL procedure 552
    ERRLIMIT=
        DBLOAD procedure 987
    ERRLIMIT= data set option 356
    ERRLIMIT= LIBNAME option 160
    error codes 446
    error files 110
    error limits
        for Fastload utility 160
        rollbacks and 356
    error messages 445
error tracking 369, 370
acquisition error tables errors 310
transcoding 228, 434
escape sequence escape sequences 280
ESCAPE_BACKSLASH= data set option 357
ESCAPE_BACKSLASH= LIBNAME option 161
turn codes SQL pass-through facility 472
exception tables 281
exclusive locks 172
EXECUTE statement SQL procedure 480
EXIT statement DB2EXT procedure 549
DB2UTIL procedure 552
explicit SQL FastExport Utility and 917
exporting tables using a map file 298
external tables accessing with protocols 595, 641
extracting data numeric precision and 9

F
Fact tables 155, 354
column name as partition key 183, 391
column names as partition key 390
FastExport retries for logging in to Teradata 212, 222, 410, 415
FastExport Utility 914
case sensitivity and 915
explicit SQL and 917
password security 915
redirecting log tables to alternate database 173
setup 916
usage 916
usage exceptions 917
versus partitioning WHERE clauses 918
FASTEXPORT= LIBNAME option 161
FASTEXPORT=data set option FASTLOAD 358
FastLoad 910
error limit for 160
examples 913

features and restrictions 910
invoking 910
retries for logging in to Teradata 212, 222, 410, 415
starting with TPT API 907
TPT API data set options 907
TPT API features and restrictions 907
FASTLOAD= BULKLOAD= data set option 317
BULKLOAD= LIBNAME option 112
FASTLOAD= data set option 360
FASTLOAD= LIBNAME option 163
FE_EXECNAME BL_CONTROL= data set option 249
features by host 77
Aster 78
DB2 under UNIX and PC Hosts 78
DB2 under z/OS 79
Greenplum 79
Hadoop 80
HAWQ 81
Impala 82
Informix 82
Microsoft SQL Server 83
MySQL 83
Netezza 84
ODBC 85
OLE DB 86
Oracle 86
PostgreSQL 87
SAP HANA 88
Sybase 88
Sybase IQ 89
Teradata 90
Vertica 90
features table 5
FETCH_IDENTITY= data set option 361
FETCH_IDENTITY= LIBNAME option 164
file allocation
bulk loading, DB2 under z/OS 562
file server configuring 596, 642
file:// protocol 597, 643
flat file path for creating for bulk loading 253
FLOAT data type
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 568
Greenplum 600
HAWQ 645
Hive (Hadoop) 616
Impala 659
Sybase 866
Sybase IQ 888
Teradata 931
| FMT statement | DB2EXT procedure | 548 |
| FORMAT | ACCESS procedure | 957 |
| FORMAT statement | ACCESS procedure | 962 |
| formats | changing from default | 962 |
| date formats | generating | 960 |
| numeric | 448 |
| FROM_LIBREF= statement | 977 |
| CV2VIEW procedure | 977 |
| FROM_VIEW= statement | 977 |
| CV2VIEW procedure | 977 |
| fully qualified table names | Informix | 678 |
| function lists, in-memory | 213 |
| FUNCTION= option | PROC BD2UTIL statement | 551 |
| functions | data functions, processing | 67 |
| LIBNAME statement and functions | 96 |
| passing SQL functions to Sybase | 861 |
| passing to Aster | 495 |
| passing to DB2 under UNIX and PC Hosts | 520 |
| passing to DB2 under z/OS | 557 |
| passing to DBMS with SQL procedure | 44 |
| passing to DBMS with WHERE clauses | 49 |
| passing to Greenplum | 594 |
| passing to Hadoop | 612 |
| passing to HAWQ | 640 |
| passing to Impala | 655 |
| passing to Informix | 672 |
| passing to Microsoft SQL Server | 691 |
| passing to MySQL | 704 |
| passing to Netezza | 719 |
| passing to ODBC | 749 |
| passing to OLE DB | 769 |
| passing to Oracle | 796 |
| passing to PostgreSQL | 819 |
| passing to SAP HANA | 839 |
| passing to Sybase | 861 |
| passing to Sybase IQ | 883 |
| passing to Teradata | 904 |
| passing to Vertica | 947 |
| SQL | 861 |
| writing to data sets or logs | 217 |
| globalization | 11 |
| transcoding errors | 228, 434 |
| gpfdist | stopping | 596, 642 |
| troubleshooting | 597, 643 |
| GRANT statement | SQL procedure | 482 |
| GRAPHIC data type | DB2 under UNIX and PC Hosts | 527 |
| DB2 under z/OS | 568 |
| Greenplum | 584 |
| accessing dynamic data in web tables | 597 |
| accessing external tables with protocols | 595 |
| autopartitioning scheme | 592 |
| binary data types | 600 |
| bulk loading | 595 |
| configuring file server | 596 |
| data conversions | 601 |
| data set options | 588 |
| data set options for bulk loading | 597 |
| data types | 599 |
| date, time, and timestamp data | 600 |
| DBSLICE= data set option | 592 |
| DBSLICEPARM= LIBNAME option | 592 |
| file:// protocol | 597 |
| LIBNAME statement | 584 |
| naming conventions | 599 |
| NULL values | 600 |
| nullable columns | 592 |
| numeric data | 600 |
| passing joins to | 595 |
| passing SAS functions to | 594 |
| special catalog queries | 591 |
| SQL pass-through facility | 590 |
| stopping gpfdist | 596 |
| string data | 599 |
| supported features | 79 |
| troubleshooting gpfdist | 597 |
| WHERE clauses | 592 |
| group ID | 242 |
| qualifying table names with | 106 |

### H

<table>
<thead>
<tr>
<th>Hadoop</th>
<th>604</th>
</tr>
</thead>
<tbody>
<tr>
<td>bulk loading</td>
<td>613</td>
</tr>
<tr>
<td>data conversion, Hive to SAS</td>
<td>617</td>
</tr>
<tr>
<td>data set options</td>
<td>610</td>
</tr>
<tr>
<td>data types</td>
<td>615</td>
</tr>
<tr>
<td>LIBNAME statement</td>
<td>605</td>
</tr>
<tr>
<td>NULL values</td>
<td>625</td>
</tr>
<tr>
<td>passing joins to</td>
<td>613</td>
</tr>
<tr>
<td>passing SAS functions to</td>
<td>612</td>
</tr>
</tbody>
</table>
SQL pass-through facility 611
supported features 80
transcoding errors 228, 434
Hadoop (Hive)
complex data 616
date, time, and timestamp data 616
simple data 616
table properties, SAS 619
Hadoop distribution
vendor 159
HAWQ 630
accessing dynamic data in web tables 643
accessing external tables with protocols 641
autopartitioning scheme 638
bulk loading 641
configuring file server 642
data conversions 646
data set options 634
data set options for bulk loading 643
data types 645
date, time, and timestamp data 646
DBSLICE= data set option 638
DBSLICEPARM= LIBNAME option 638
file:// protocol 643
LIBNAME statement 630
naming conventions 644
NULL values 646
nullable columns 638
numeric data 645
passing joins to 640
passing SAS functions to 640
special catalog queries 637
SQL pass-through facility 636
stopping gpfdist 642
string data 645
supported features 81
troubleshooting gpfdist 643
WHERE clauses 638
HDFS location, specifying
Hive tables 331
HDFS_DATADIR= connection option 607
HDFS_KERBEROS_PRINCIPAL= HDFS_PRINCIPAL= LIBNAME option 165
HDFS_METADIR= connection option 608
HDFS_PRINCIPAL= connection option 608
HDFS_PRINCIPAL= data set option 362
HDFS_PRINCIPAL= LIBNAME option 165
HDFS_TEMPDIR= connection option 643
Hadoop 608
heterogeneous joins
pushing to DBMS 41
Hive (Hadoop)
converting data to SAS 618
data conversion from SAS to Hive (Hadoop) 619
existing table properties 620
naming conventions 615
table properties, SAS 619
Hive (Hadoop) numeric data types 616
Hive to SAS (Hadoop)
data conversion 617
HIVE_PRINCIPAL= connection option 608
host name
bulk loading 108
for SAP HANA server 311
host, features by 77
HOST=
SERVER= connection option for Hadoop 607
SERVER= connection option for Impala 651
SERVER= connection option for SAP HANA 829
HOST= connection option
Sybase IQ 874
I
identifiers
deliminating 199
identity column 294
populating during bulk loading 109
identity values 361
last inserted 164
IGNORE_READ_ONLY_COLUMNS= data set option 362
IGNORE_READ_ONLY_COLUMNS= LIBNAME option 165
Impala 233, 650
bulk loading 656
bulk-loading options 117, 118, 323, 324
configuration 117, 118
configuration directory 323, 324
connection options 654
data set options 653
LIBNAME statement 650
naming conventions 658
passing joins to 656
passing SAS functions to 655
SQL pass-through connection options 654
SQL pass-through facility 654
supported features 82
Impala data types 659
Impala date and time data types 659
Impala numeric data types 659
Impala string data types 659
Impala to SAS data conversions 660
IMPALA_PRINCIPAL= LIBNAME option 167
IMPORT statement
  additional options 290
importing
  table data accessible during import 243
in-database procedures
  generating SQL for 465
in-memory function lists 213
IN= data set option 364
IN= LIBNAME option 167
IN= option
  PROC DB2EXT statement 548
indexes 335
maintenance, DB2 load utility 293
processing joins of large table and small data set 139
Sybase 856
Informix 664
  autopartitioning scheme 671
  character data 674
data conversions, LIBNAME statement 676
data conversions, SQL pass-through facility 677
data set options 667
data types 674
database servers 677
date, time, and interval data 675
DBDATASRC environment variables 678
DBSLICE= data set option 672
DBSLICEPARM= LIBNAME option 671
default environment 664
fully qualified table names 678
LIBNAME statement 664
locking 673
naming conventions 674
numeric data 675
online database servers 677
passing joins to 673
passing SAS functions to 672
SE database servers 677
servers 677
SQL pass-through facility 668
stored procedures 669
supported features 82
temporary tables 672
WHERE clauses 671
Informix stored procedures
  SQL command restrictions for 669
INITCMD
  DBCONINIT= LIBNAME option 133
initialization command
  executing after every connection 133
  executing once 140
  user-defined 133, 140, 475
InnoDB table type 703
input processing
  overriding default SAS data types 346
insert processing
  forcing truncation of data 333
INSERT statement
  SQL procedure 482
insert statements
  Teradata 178, 379
INSERT_SQL= data set option 364
INSERT_SQL= LIBNAME option 168
INSERTBUFF= data set option 365
INSERTBUFF= LIBNAME option 169
inserting data
  appending data sets to DBMS tables 997
  DB2 tables 549, 553
  limiting observations loaded 987
  loading data subsets into DBMS tables 987
  saving DBMS table after inserts 987
installation 65
INSTANCE= connection option
  SAP HANA 829
INT data type
  Hive (Hadoop) 616
  Impala 659
  Sybase 866
INT_STRING= connection option
  OLE DB 758
INTEGER data type
  casting 114
  DB2 under UNIX and PC Hosts 527
  DB2 under z/OS 568
  Greenplum 600
  HAWQ 645
  Netezza 724
  Sybase IQ 888
  Teradata 931
INTERFACE= database description statement
  Sybase 860
INTERFACE= LIBNAME option 170
interfaces 65
  features by host 77
  invoking 65
  threaded Reads and 54
interfaces file
  name and location of 170
intermediate files for bulk loading 107
internationalization transcoding errors 228, 434
interrupting SQL processes 179, 226
INTERVAL DAY TO SECOND data type
 Oracle 803
INTERVAL YEAR TO MONTH data type
 Oracle 803
IP address of SAP HANA server 311
IP_CURSOR= connection option
 Sybase 851
isolation levels 201, 228, 436

J
JDBC
 for reading data 203, 402
JDBC connection options
 PROPERTIES= LIBNAME option 195
joins
determining larger table 340
failures when passing to DBMS 47
indexes for joins of large table and small data set 139
outer joins 217
passing to Aster 496
passing to DB2 under UNIX and PC Hosts 521
passing to DB2 under z/OS 558
passing to DBMS 45
passing to Greenplum 595
passing to Hadoop 613
passing to HAWQ 640
passing to Impala 656
passing to Informix 673
passing to MySQL 705
passing to Netezza 720
passing to ODBC 750
passing to OLE DB 770
passing to Oracle 797
passing to PostgreSQL 820
passing to SAP HANA 840
passing to Sybase 862
passing to Sybase IQ 884
passing to Teradata 905
passing to Vertica 948
performance of joins between two data sources 176
performed by SAS 50
processing 67
pushing heterogeneous joins 41

K
KEEP= data set option
limiting retrieval 38
Kerberos
 principal name 165, 167
key column for DBMS retrieval 337
keyset-driven cursor 170
KEYSET_SIZE= data set option 366
KEYSET_SIZE= LIBNAME option 170
keysets 366
 number of rows driven by 170

L
LABEL
 DBLOAD procedure 987
labels
column labels for engine use 148
DBMS column names defaulting to 987
language differences
 See NLS
language support
 Sybase 871
large numeric values
 loading in Netezza 724
Large object (LOB) data
 DB2 under z/OS 569
last inserted identity value 164
length
 CHAR or VARCHAR data type
columns 102
column length for client encoding 131
fixed-width column length for client encoding 129
fixed-width column length for database server encoding 148
names 14
NCHAR or NVARCHAR data type
columns 103
very long character data types 142
LIBNAME options 97, 101
 for NLS 63
MultiLoad with TPT API 908
prompting window for specifying 100
LIBNAME statement
See SAS/ACCESS LIBNAME statement
libraries
 containing descriptors for conversion 977
disassociating librefs from 99
writing attributes to log 99
librefs
 assigning interactively 97
assigning to remote DBMS 101
assigning with LIBNAME statement 96, 100
client-server authorization and 575
DBMS connections and disassociating from libraries level for opening connections pointing to database servers shared connections for multiple librefs

LIMIT= DBLOAD procedure LIMIT= option PROC DB2UTIL statement links database links from default database to another database from local database to database objects on another server

LIST ACCESS procedure DBLOAD procedure LIMIT= option LIBNAME statement LIST statement ACCESS procedure literals backslashes in LOAD DBLOAD procedure Load operator load performance examples FastLoad and MultiLoad and Teradata LOAD process recoverability of LOAD utility base filename and location of temporary files control statement execution mode index maintenance restarts running against existing tables SYSDISC data set name SYSSIN data set name SYSPARAM data set name SYSPRINT data set name SYSPRINT data set name SYSREC data set name temporary data sets unique identifier for a given run loading data error limit for Fastload utility loading tables bulk copy for using a map file LOC=

LOCATION= data set option LOCATION= LIBNAME option local databases linking to database objects on another server LOCATION= connection option DB2 under z/OS LOCATION= data set option LOCATION= LIBNAME option locking controlling exclusive locks Informix Microsoft SQL Server ODBC OLE DB Oracle PostgreSQL Sybase Sybase IQ Teradata wait time for unlock time for LOCKTABLE= data set option LOCKTABLE= LIBNAME option LOCKTIME= LIBNAME option LOCKWAIT= LIBNAME option log writing functions to writing library attributes to log files for bulk loading log tables redirecting to alternate database LOGDBC= LIBNAME option login timeout LOGIN_TIMEOUT= LIBNAME option long DBMS data type

LONG VARCHAR data type DB2 under UNIX and PC Hosts DB2 under z/OS Sybase IQ Teradata

LONG VARGRAPHIC data type DB2 under UNIX and PC Hosts DB2 under z/OS
M

macro variables 445
capturing bulk-load statistics into 523
management class for SMS-managed data set 263
MAP data type
   Hive (Hadoop) 616
map file 298
MAPTO statement
   DB2UUTIL procedure 551
MAX_CONNECTS= LIBNAME option 175
maximum bytes
   for database server encoding 150
MBUFFSIZE= data set option 368
   MultiLoad and 912
MDX command
   defining result data sets 115
MDX statements 772
memory
   for bulk loading 252
metadata
   for result data sets 115
metadata, character set
   transfer from input to output for processing 207
Microsoft Bulk Copy (BCP) facility 747, 750
Microsoft SQL Server 681
   configuring partitioned views for
      DBSLICE= 744
data conversions 693
data set options 686
data types 693
DBLOAD procedure 690
LIBNAME statement 682
locking 691
naming conventions 693
NULL values 693
NULL values and bulk loading 109, 294
passing SAS functions to 691
populating identity column during bulk loading 109
SQL pass-through facility 688
supported features 83
missing values 299, 381
   replacing character values 382
result set differences and 33
ML_CHECKPOINT= data set option 368, 912
ML_ERROR1= data set option 369, 912
ML_ERROR2= data set option 370, 912
ML_LOG= data set option 371, 912
ML_RESTART= data set option 372, 912
ML_WORK= data set option 373, 912
MOD function
   autopartitioning and 59
differences between SAS and DBMS 44
MOD partitioning
   column selection for 541
MODE= LIBNAME option 175
MONEY data type
   Sybase 866
multi-statement insert
   starting with TPT API 909
TPT API data set options 909
TPT API features and restrictions 909
MULTI_DATASRC_OPT= data set option
   joins and 50
MULTI_DATASRC_OPT= LIBNAME option 176
MultiLoad 911
   acquisition error tables 369, 370
   bulk loading 374
data buffers 374
data set options 911
   enabling/disabling 373
   examples 378, 913
   features and restrictions 911
   prefix for temporary table names 371
   restart table 372
   restarting 375
   retries for logging in to Teradata 212, 222, 410, 415
   setup 911
   starting with TPT API 908
   storing intermediate data 373
temporary tables 374
TPT API data set options 908
TPT API features and restrictions 908
TPT API LIBNAME options 908
work table 373
MULTILOAD= data set option 373
   multiphase commit and rollback calls 559
MULTISTMT= data set option 379
MULTISTMT= LIBNAME option 178
MyISAM table type 703
MySQL 697
   autocommit and table types 703
   bulk loading 705
   case sensitivity 707
   character data 707
   data conversions 708
data set options 700
data types 707
   date, time, and timestamp data 707
   LIBNAME statement 698
   naming conventions 706
   numeric data 707
   passing functions to 704
passing joins to  705
SQL pass-through facility  702
supported features  83
update and delete rules  703

N
name literals  15, 24
named pipes  315
names 13
See also naming conventions
ACCESS procedure  15
behavior when creating DBMS objects  18
behavior when retrieving DBMS data  17
database name for bulk loading  271
DB2 under z/OS, bulk loading file allocation  562
DBLOAD procedure  16
DBMS columns  987
DBMS tables  987
default behaviors  15
double quotation marks and  17
elements of  19
length of  14
modification and truncation  15
name literals  15, 24
options affecting  17
overriding naming conventions  17
preserving column names  20
preserving table names  20
renaming DBMS columns to valid SAS names  138
renaming DBMS data  16
replacing unsupported characters  19
renaming conventions  14
See also names
DB2 under UNIX and PC Hosts  526
DB2 under z/OS  567
Greenplum  599
HAWQ  644
Hive (Hadoop)  615
Impala  658
Informix  674
Microsoft SQL Server  693
MySQL  706
ODBC  752
OLE DB  775
Oracle  801
PostgreSQL  822
SAP HANA  841
Sybase  864
Sybase IQ  887
Teradata  929
national language support
Sybase  871
National Language Support  11
See also NLS
NCHAR data type
Netezza  724
NCHAR data type columns
adjusting default for  116
adjusting lengths for  103
NCHAR or NVARCHAR data type columns  116
Netezza  711
bulk loading and unloading  721
connection options  718
data conversions  725
data set options  716
data types  723
date, time, and timestamp values  724
LIBNAME statement  712
NULL values  725
passing joins to  720
passing SAS functions to  719
rapidly retrieving a large number of rows  112
special catalog queries  718
SQL pass-through connection options  718
SQL pass-through facility  718
supported character data types  724
supported features  84
supported numeric data types  724
temporary tables  719
network name of remote host  311
New Library window  97
NLS
data types  64
options  63
Sybase  871
NLS (National Language Support)  63
NODB2RRS system option  559
NODB2RRSMP system option  559
non-ANSI standard SQL  4
nonrepeatable reads  525
Microsoft SQL Server  692
ODBC  751
OLE DB  772
PostgreSQL  822
Sybase IQ  886
Vertica  950
NOPROMPT= connection option
Aster  487
DB2 under UNIX and PC Hosts  505
Microsoft SQL Server  683
ODBC  733
SAP HANA  830
null value string  299
NULL values
accepted in DBMS columns 987
as valid value when tables are created 342
bulk loading and 109
DB2 under UNIX and PC Hosts 528
DB2 under z/OS 569
Greenplum 600
Hadoop 625
HAWQ 646
Microsoft SQL Server 693
Microsoft SQL Server columns 294
Netezza 725
ODBC 753
Oracle 806
PostgreSQL 823
result set differences and 33
SAP HANA 843
Sybase 867
Sybase IQ 888
Teradata 932
Vertica 952
nullable columns
Aster 494
DB2 under UNIX and PC Hosts 513
Greenplum 592
HAWQ 638
ODBC 743
SAP HANA 838
Sybase IQ 882
NULLCHAR= data set option 381
NULLCHARVAL= data set option 382
NULLIF clause
 suppressing 314
NULLS
 DBLOAD procedure 987
 NULLS statement, DBLOAD procedure 748
 NUMBER data type
 Oracle 802
 NUMBER(p,s) data type
 Oracle 802
 NUMBER(p) data type
 Oracle 802
numeric data
Aster 499
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 568
Greenplum 600
HAWQ 645
Informix 675
MySQL 707
Netezza 724
Oracle 802
SAP HANA 842
Sybase IQ 888
Vertica 951
NUMERIC data type
 DB2 under UNIX and PC Hosts 527
 Greenplum 600
 HAWQ 645
 Netezza 724
 Sybase 866
 Sybase IQ 888
numeric data types
 Hive (Hadoop) 616
 Impala 659
 Teradata 931
Numeric data types
Sybase 866
numeric formats 448
numeric precision 7
data representation and 8
options for choosing degree of 10
references for 11
rounding and 8
selectively extracting data and 9
numeric table columns
specifying the default data type 178
numeric values
 precision 724
NVARCHAR data type
Netezza 724
NVARCHAR data type columns
adjusting default for 116
adjusting lengths for 103

O
objects
 naming behavior when creating 18
objects, database 129
observations 987
ODBC 728
autopartitioning scheme 743
bulk loading 750
components and features 728
Cursor Library 233
data conversions 753
data set options 737
data types 752
DBLOAD procedure 747
DBSLICE= data set option 744
DBSLICEPARM= LIBNAME option 744
LIBNAME statement 732
locking 750
naming conventions 752
NULL values 753
nullable columns 743
passing joins to 750
passing SAS functions to 749
PC platform 728, 730
special catalog queries 741
SQL pass-through facility 738
SQL Server partitioned views for
DBSLICE= 744
supported features 85
temporary tables 749
UNIX platform 729, 730
WHERE clauses 743
OLAP data
accessing with OLE DB 772
OLE DB SQL pass-through facility with 773
OLE DB 755
accessing OLAP data 772
Boolean environment variable 776
bulk loading 771
connecting directly to data provider 761
connecting with OLE DB services 761
data conversions 776
data set options 762
data types 775
LIBNAME statement 756
locking 771
naming conventions 775
passing joins to 770
passing SAS functions to 769
special catalog queries 765
SQL pass-through facility 764, 773
supported features 86
temporary tables 769
OLE DB services
connecting with 761
OLEDB_SERVICES= connection option 757
open database connectivity
See ODBC
operating system command for segment instances 282
optimizing SQL usage
See SQL usage, optimizing options
affecting naming behavior 17
bulkload 290
for NLS 63
SFTP configuration 312
OR_BINARY_DOUBLE= LIBNAME option 178
OR_ENABLE_INTERRUPT=
LIBNAME option 179
OR_IDENTITY_COLS= data set option 382
OR_PARTITION= data set option 383
OR_UPD_NOWHERE= data set option 386
OR_UPD_NOWHERE= LIBNAME option 180
Oracle 780
ACCESS procedure 792
autopartitioning scheme 789
binary data 803
bulk loading 295, 798
CHAR and VARCHAR2 column lengths 128
character data 802
data conversions, ACCESS procedure 809
data conversions, DBLOAD procedure 810
data conversions, LIBNAME statement 807
data set options 784
data types 802
database links 339
date, timestamp, and interval data 803
DBLOAD procedure 794
hints 387
LIBNAME statement 780
linking from local database to database objects on another server 142
locking 800
naming conventions 801
nonpartitioned tables 791
NULL and default values 806
numeric data 802
partitioned tables 790
passing joins to 797
passing SAS functions to 796
performance 792, 796
SQL pass-through facility 787
supported features 86
Oracle SQL*Loader
See SQL*Loader
ORACLE_73_OR_ABOVE=
OR_UPD_NOWHERE= data set option 386
OR_UPD_NOWHERE= LIBNAME option 180
ORAPW=
PASSWORD= connection argument for CONNECT statement 788
ordering DBMS data 328
ORHINTS= data set option 387
OUT= option
PROC ACCESS statement 959
PROC DB2EXT statement 548
outer joins 217
overhead limit for data conversions 115
for data conversions in Teradata instead of SAS 321
OVERLAPS= data set option
overlapping time periods 388
P
packet size 180
PACKETSIZE= LIBNAME option 180
parallelism 353
building table objects 251
for DB2 154, 353
writing data to disk 279
PARMDEFAULT= data set option 389
PARMDEFAULT= LIBNAME option 181
PARMSTRING= data set option 389
PARMSTRING= LIBNAME option 181
PARTITION
PARTITIONED_BY= data set option 391
partition key
creating fact tables 183, 391
PARTITION_BY
PARTITIONED_BY= data set option 391
PARTITION_KEY= data set option 390, 391
PARTITION_KEY= LIBNAME option 183
partitioned tables
Oracle 790
partitioned views
SQL server, configuring for DBSLICE= 744
PARTITIONED_BY= data set option 391
partitioning 383
See also autopartitioning
column selection for MOD partitioning 541
queries for threaded Reads 347
partitioning WHERE clauses
FastExport versus 918
threaded reads 917
pass-through facility
See SQL pass-through facility
pass-through functions
MOD 44
PASS=
PASSWORD= connection option 812
PASSWORD= connection option for Aster 487
PASSWORD= connection option for Hadoop 607
PASSWORD= connection option for Impala 651
PASSWORD= connection option for Netezza 713
PASSWORD= connection option for Sybase 851
passing functions, with WHERE clauses 49
joins 45
joins, and failures 47
SQL DELETE statement to empty a table 46
WHERE clauses 49
PASSWORD=
USING= connection option for Informix 665
PASSWORD= connection argument
CONNECT statement 788
PASSWORD= connection option
Aster 487
DB2 under UNIX and PC Hosts 505
Greenplum 585
Hadoop 607
HAWQ 631
Impala 651
Microsoft SQL Server 682
MySQL 698
Netezza 713
ODBC 732
OLE DB 756
Oracle 781
PostgreSQL 812
SAP HANA 830
Sybase 851
Sybase IQ 875
Teradata 894
Vertica 941
PASSWORD= database description statement
Sybase 859
PASSWORD= statement, DBLOAD procedure
Microsoft SQL Server 690
ODBC 747
passwords
assigning 28
data set and descriptor access 971
FastExport Utility 915
protection levels 28
path
for bulk loading 302
PATH= connection option
Oracle 781
PC Hosts
See DB2 under UNIX and PC Hosts
PC platform
ODBC on 728, 730
performance
DB2 under z/OS 554
increasing SAS server throughput 37
indexes for processing joins 139
joins between two data sources 176
limiting retrieval 37
optimizing SQL statements in pass-through facility 449
optimizing SQL usage 43
Oracle 792, 796
processing queries, joins, and data functions 67
reducing table read time 53
repeatedly accessing data 39
sorting DBMS data 39
temporary table support 40
Teradata load performance 905
threaded Reads and 53, 58
permanent tables 143
phantom reads
  DB2 under UNIX and PC Hosts 526
  Microsoft SQL Server 692
  ODBC 751
  OLE DB 772
  Sybase IQ 886
physical database connections sharing 121
physically partitioned databases configuring EEE nodes on 515
pipes 315
plans
  for connecting or binding SAS to DB2 559
port numbers 111, 303, 304
PORT= connection option
  Aster 487
  Greenplum 584
  Hadoop 607
  HAWQ 631
  Impala 651
  Netezza 712
  PostgreSQL 812
  SAP HANA 829
  Sybase IQ 875
  Vertica 940
POST_STMT_OPTS= data set option 394
PRE_STMT_OPTS= data set option 394
PRE_TABLE_OPTS= data set option 395
precision, numeric
  See numeric precision
  preserve case sensitivity of user name 193
  PRESERVE_COL_NAMES= data set option 397
  naming behavior and 17
  PRESERVE_COL_NAMES= LIBNAME option 185
  naming behavior and 17
  PRESERVE_GUID= LIBNAME option 186
  PRESSERVE_NAMES=
    PRESERVE_COL_NAMES= data set option 397
    PRESERVE_COL_NAMES= LIBNAME option 185
    PRESERVE_TAB_NAMES= LIBNAME option 191
    naming behavior and 17
    PRESERVE_USER= LIBNAME option 193
  preserving
    column names 20
    table names 20
    privileges 27, 560
PROC ACCESS
  ACCESS procedure 957
  PROC CV2VIEW statement 976, 977
  PROC DBLOAD statement 0
  procedures
    Informix stored procedures 669
  PROGRAM_NAME= LIBNAME option 194
PROMPT= connection option
  Aster 487
  DB2 under UNIX and PC Hosts 505
  Microsoft SQL Server 683
  ODBC 733
  OLE DB 757
  SAP HANA 830
  prompting window 100
  prompts 475
  DBMS connections and 344
  to enter connection information 146
  PROPERTIES= connection option
  OLE DB 757
  PROPERTIES= LIBNAME option 195
  protocols
    accessing external tables 595, 641
    for bulk loading 305
PROVIDER_STRING= connection option
OLE DB 757
PROVIDER= connection option
OLE DB 756
PSQL executable file location specification 306
pushing heterogeneous joins 41
updates 42
PUT function mapping to SAS_PUT function 468
PW=
PASSWORD= connection option 812
PASSWORD= connection option for Aster 487
PASSWORD= connection option for Hadoop 607
PASSWORD= connection option for Impala 651
PASSWORD= connection option for Netezza 713
PASSWORD= connection option for Sybase 851
PWD=
PASSWORD= connection option 812
PASSWORD= connection option for Aster 487
PASSWORD= connection option for Hadoop 607
PASSWORD= connection option for Impala 651
PASSWORD= connection option for Microsoft SQL Server 682
PASSWORD= connection option for Netezza 713
PASSWORD= connection option for ODBC 732
PASSWORD= connection option for Vertica 941
USING= connection option for Informix 665

HAWQ 637
Netezza 718
ODBC 741
OLE DB 765
ordering results with BY clause 919
partitioning for threaded Reads 347
processing 67
retrieving and using DBMS data in Sybase IQ 880
timeout for 198, 399
query bands 197, 399
query results locations for storing 207
QUERY_BAND= data set option 399
QUERY_BAND= LIBNAME option 197
QUERY_TIMEOUT= data set option 399
QUERY_TIMEOUT= LIBNAME option 198
QUIT
ACCESS procedure 957
DBLOAD procedure 987
quotation character for CSV mode 248, 306
quotation marks delimiting identifiers 199
double 17
QUOTE_CHAR= LIBNAME option 199
QUOTED_IDENTIFIER= LIBNAME option 199

R
random access engine
SAS/ACCESS engine as 206
RAW data type
Oracle 803
read-only columns 362
ignoring when generating SQL statements 165
read-only cursors 125
READ_ISOLATION_LEVEL= data set option 401
READ_ISOLATION_LEVEL= LIBNAME option 201
READ_LOCK_TYPE= data set option 401
READ_LOCK_TYPE= LIBNAME option 202
READ_METHOD= data set option 402
READ_METHOD= LIBNAME option 203
READ_MODE_WAIT= data set option 403
READ_MODE_WAIT= LIBNAME option 204
READ_ONLY= connection option

Q
QUALIFIER= data set option 398
QUALIFIER= LIBNAME option 196
qualifiers reading database objects 398
QUALIFY_ROWS= LIBNAME option 197
qualifying table names 106
queries Aster 493
DB2 under UNIX and PC Hosts 512
Greenplum 591

Index 1037
NETEZZA 713
READBUFF= connection option  
ODBC 733
READBUFF= data set option 400
READBUFF= LIBNAME option 200
reading data 68
with TPT API 161
READONLY=  
READ_ONLY= connection option for Netezza 713
REAL data type  
DB2 under z/OS 568
Greenplum 600
HAWQ 645
Netezza 724
Sybase 866
Sybase IQ 888
records
rejected records 245, 284
to insert for commit 290
Recoverable Resource Manager Services  
Attachment Facility (RRSAF) 559, 575, 578, 580
reject limit count 308
rejected records 245, 284
relational databases
access methods 4
selecting an access method 4
remote DBMS
assigning libref to 101
remote host
network name 311
remote library services (RLS) 101
remote stored procedures 518, 544
REMOTE_DBTYPE= LIBNAME option 205
RENAME  
ACCESS procedure 957
DBLOAD procedure 987
RENAME statement  
ACCESS procedure 964
DB2EXT procedure 548
renaming
columns 16, 334, 987
DBMS data 16
tables 16
variables 16, 964
repeatable reads
Informix 673
repeatedly accessing data 39
REPLACE= data set option 241
REPLACE= statement 978
CV2VIEW procedure 978
representing data
numeric precision and 8
REQUIRED= connection option
ASTER 487
DB2 under UNIX and PC Hosts 505
Microsoft SQL Server 683
ODBC 733
OLE DB 758
REREAD_EXPOSURE= LIBNAME option 206
RESET  
ACCESS procedure 957
DBLOAD procedure 987
RESET statement  
ACCESS procedure 966
DB2UTIL procedure 551
Resource Limit Facility (DB2 under z/OS) 554
restart table 372
restrictions (command) for SQL
for Informix stored procedures 669
result sets
metadata and content of 115
null data and 33
qualifying member values 197
RESULTS= LIBNAME option 207
retain case sensitivity of user name 193
retrieving data
KEEP= and DROP= options for limiting 38
limiting retrieval 37
naming behavior and 17
row and column selection for limiting 37
return codes 445
DB2 under z/OS 560
REVOKE statement  
SQL procedure 482
rollbacks
error limits and 356
rounding data
numeric precision and 8
ROW_DELIM
ROW_DELIMITER= data set option 404
ROW_DELIMIT
ROW_DELIMITER= data set option 404
ROW_DELIMITER= data set option 404
ROWID data  
DB2 under z/OS 569
rows
DB2 tables 549
deleting multiple rows 155
distributing across database segments 355
duplicate 409
inserting 168, 364
limiting retrieval 37

number driven by keyset 170
number in single insert operation 169, 365
number to process 232
rapidly retrieving 112
reading into buffers 200, 400
updating and deleting in data sources 231
updating with no locking 180
wait time before locking 172
waiting indefinitely before locking 173
ROWSET_SIZE=
READBUFF= connection option for ODBC 733
READBUFF= data set option 400
READBUFF= LIBNAME option 200
ROWSET=
READBUFF= data set option 400
READBUFF= LIBNAME option 200
RRSAF (Recoverable Resource Manager Services Attachment Facility) 559, 575, 578, 580

S
sampling
Teradata 922
SAP HANA 828
autopartitioning scheme 838
binary string data 842
bulk loading 841
data conversions 846
data set options 834
data types 842
date, time, and timestamp data 842
DBSLICE= data set option 838
DBSLICEPARAM= LIBNAME option 838
LIBNAME statement 828
naming conventions 841
NULL values 843
nullable columns 838
numeric data 842
passing joins to 840
passing SAS functions to 839
server name 311
SQL pass-through facility 836
string data 842
supported features 88
update and delete rules 837
WHERE clauses 838
SAS data types 659
SAS data views 6
SAS functions
passing to Aster 495
passing to DB2 under UNIX and PC
Hosts 520
passing to DB2 under z/OS 557
passing to Greenplum 594
passing to Hadoop 612
passing to HAWQ 640
passing to Impala 655
passing to Informix 672
passing to Microsoft SQL Server 691
passing to MySQL 704
passing to ODBC 749
passing to OLE DB 769
passing to Oracle 796
passing to Sybase 861
passing to Sybase IQ 883
passing to Teradata 904
SAS security 28
assigning passwords 28
controlling DBMS connections 31
customizing DBMS connect and disconnect exits 33
defining views and schemas 31
extracting DBMS data to data sets 30
locking, transactions, and currency control 32
protecting connection information 30
securing data 28
SAS server
increasing throughput 37
SAS table properties
Hive (Hadoop) 619
SAS to Hive (Hadoop)
data conversion 619
SAS to Impala data conversions 661
SAS views
creating 7
SAS_DBMS_AUTOMETADATA=
LIBNAME option 207
SAS_PUT() function
mapping PUT function to 468
SAS/ACCESS
features by host 77
installation requirements 65
interactions with SQL procedure 471
interfaces 65
interfaces and threaded Reads 54
invoking interfaces 65
names 13
National Language Support 11
task table 5
SAS/ACCESS engine
as random access engine 206
blocking operations and 453
buffering bulk rows for output 317
reading data with TPT API 161
SAS/ACCESS LIBNAME statement 4, 96
accessing data from DBMS objects 66
advantages of 4
alternative to 471
arguments 97
assigning librefs 96, 100
assigning librefs interactively 97
assigning librefs to remote DBMS 101
Aster 486
Aster data conversions 500
connection options 97
data from a DBMS 99
DB2 under UNIX and PC Hosts 504
DB2 under UNIX and PC Hosts data conversions 528
DB2 under z/OS 533
DB2 under z/OS data conversions 570
DBMS specifics 96
disassociating librefs from libraries 99
functions and 96
Greenplum 584
Greenplum data conversions 601
Hadoop 605
HAWQ 630
HAWQ data conversions 646
how it works 66
Impala 650
Informix 664
Informix data conversions 676
LIBNAME options 97, 101
Microsoft SQL Server 682
Microsoft SQL Server data conversions 693
MySQL 698
MySQL data conversions 708
Netezza 712
Netezza data conversions 725
ODBC 732
ODBC data conversions 753
OLE DB 756
OLE DB data conversions 776
Oracle 780, 807
PostgreSQL 812
PostgreSQL data conversions 823
prompting window and LIBNAME options 100
SAP HANA 828
SAP HANA data conversions 846
sorting data 96
SQL views embedded with 99
Sybase 850
Sybase data conversions 867
Sybase IQ 874
Sybase IQ data conversions 889
syntax 97
Teradata 893, 932
TPT API LIBNAME options 906
Vertica 940
writing library attributes to log 99
SAS/ACCESS views 7
SASDATEFMT= data set option 404
SASTRACE= system option 455
location of trace messages 464
SASTRACELOC= system option 464
SAVEAS= statement 978
CV2VIEW procedure 978
scale
for timestamp literal 127, 326
SCHEMA=
AUTHID= data set option 242
AUTHID= LIBNAME option 106
SCHEMA= connection option
Hadoop 607
Impala 651
Teradata 895
SCHEMA= data set option 406
SCHEMA= LIBNAME option 208
schemas 208, 406
data security 31
for stored procedures 518, 544
security 28
See also data security
See also SAS security
assigning passwords 28
DBMS 27
Kerberos 165, 167
privileges 27
result set differences and null data 33
SAS 28
securing data 28
service principal name (SPN) 165, 167,
362
triggers 28
segment host access
file location on Web server for 296
segment instances 282
operating system command for 282
SEGMENT_NAME= data set option 408
segments
creating tables in 408
SELECT
ACCESS procedure 957
SELECT statement
ACCESS procedure 967
DB2EXT procedure 549
server encoding
maximum bytes for 150
maximum bytes per single character 150
server name
SFTP 311
SERVER=
TDPID= connection option for Teradata

SERVER= connection option
Aster 486
DB2 under z/OS 534
Greenplum 584
Hadoop 607
HAWQ 630
Impala 651
Informix 665
MySQL 698
Netezza 712
PostgreSQL 812
SAP HANA 829
Sybase 851
Sybase IQ 874
Vertica 940
SERVER= database description statement
Sybase 860
SERVERNODE=
SERVER= connection option for SAP HANA 829
servers
connecting with name of authentication domain metadata object 105
Informix 677
SESSIONS= data set option 408
SESSIONS= LIBNAME option 210
SET= data set option 409
SFTP
options 312
SFTP access
user name 312
shared locks 172
sharing physical DBMS connections 121
SHOW_SYNONYMS=
SYNONYMS= LIBNAME option 219
SHOW_SYNONYMS= LIBNAME option 211
simple data
Hive (Hadoop) 616
simultaneous connections
maximum number allowed 175
SLEEP= data set option 212, 410
MultiLoad and 912
SLEEP= LIBNAME option 212
SLHOSTNAMEINCERT=
SLHOSTNAMEINCERTIFICATE=
connection option for SAP HANA 831
SMALLDATETIME data type
Sybase 866
SMALLINT data type
casting 114
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 568
Greenplum 600
HAWQ 645
Hive (Hadoop) 616
Impala 659
Netezza 724
Sybase 866
Sybase IQ 888
Teradata 931
SMALLMONEY data type
Sybase 866
SMS-managed data data set
data class for 258
SMS-managed data set
management class for 263
storage class for 268
SMS-managed data sets
data class for 258
SORT procedure
replacing with BY clause 921
Teradata 921
sorting data 96
performance and 39
subsetting and ordering DBMS data 328
threaded Reads and data ordering 60
source data
generating SQL for in-database processing of 465
special catalog queries
Aster 493
DB2 under UNIX and PC Hosts 512
Greenplum 591
HAWQ 637
Netezza 718
ODBC 741
OLE DB 765
Sybase IQ 880
special characters
stored in SYSDBMSG macro 446
spool files 212
SPOOL= LIBNAME option 212
SQL
ANSI-standard 4
codes to ignore (DB2) 105
DBLOAD procedure 987
executing statements 480
generating for in-database processing of source data 465
interrupting processes on DBMS server 179
interrupting processes on Teradata server 226
non-ANSI standard 4
optimizing statement handling 449
passing delete statements 156
passing generated SQL to DBMS for processing 157
SQL functions
passing to Sybase 861
SQL pass-through connection options
Impala 654
Netezza 718
SQL pass-through facility 4, 471
advantages of 5
Aster 492
CONNECT statement 473
connecting with DBMS 473
CONNECTION TO component 477
DB2 under UNIX and PC Hosts 511
DB2 under z/OS 539
DBMS specifics 472
DISCONNECT statement 480
EXECUTE statement 480
generated return codes and error messages 446
Greenplum 590
Hadoop 611
HAWQ 636
how it works 67
Impala 654
Informix 668
Informix data conversions 677
Microsoft SQL Server 688
MySQL 702
Netezza 718
ODBC 738
OLE DB 764, 773
optimizing statement handling 449
Oracle 787
PostgreSQL 817
retrieving and using DBMS data in SQL queries or views 477
return codes 472
SAP HANA 836
sending statements to DBMS 480
shared connections for multiple CONNECT statements 124
Sybase 855
Sybase IQ 880
syntax 472
tasks completed by 472
Teradata 901
terminating DBMS connections 480
Vertica 945
SQL pass-through options
for NLS 63
SQL procedure
CONNECT statement 473
creating tables 24
DISCONNECT statement 480
EXECUTE statement 480
interactions with SAS/ACCESS 471
passing DELETE statement to empty a table 46
passing functions to DBMS 44
specifying data set options 242
values within double quotation marks 17
SQL statement
DB2UTIL procedure 552
SQL usage, optimizing 43
DBINDEX=, DBKEY=, and
MULTI_DATASRC_OPT= options 50
failures of passing joins 47
passing DELETE statement to empty a table 46
passing DISTINCT and UNION processing 48
passing functions to DBMS 44
passing joins 45
passing WHERE clauses 49
SQL views 7
embedded LIBNAME statements in 99
retrieving and using DBMS data in 477
SQL_FUNCTIONS_COPY= LIBNAME option 217
SQL_FUNCTIONS= LIBNAME option 213
SQL_OJ_ANSI= LIBNAME option 217
SQL*Loader 798
blank spaces in CHAR/V ARCHAR2 columns 305
command line options 301
DIRECT option 277
discarded rows file 278
index options for bulk loading 292
z/OS 799
SQLGENERATION= LIBNAME option 218
SQLGENERATION= system option 465
SQLIN= option
PROC DB2UTIL statement 551
SQLLDR executable file location specification 313
SQLLDR_INDEX_OPTION= BL_INDEX_OPTIONS= data set option 292
SQLMAPPUTTO= system option 468
SQLOUT= option
PROC DB2UTIL statement 551
SQLXMSG macro variable 445
SQLXRC macro variable 445
SSID= connection option
DB2 under z/OS 534
SSID= option
PROC DB2EXT statement 548
PROC DB2UTIL statement 551
SSLCREATECERT= SSLCREATESELFSGNEDCERTIFICATE=
ATE= = connection option for SAP HANA 831
SSLVALIDATECERTIFICATE= connection option
SAP HANA 831
ST GEOMETRY data type
Netezza 724
statistics
capturing bulk-load statistics into macro variables 523
storage class for SMS-managed data set 268
storage location for query results 207
stored procedures basic all 542
basic call 516
DB2 under UNIX and PC Hosts 516
DB2 under z/OS 542
Informix 669
passing NULL parameter 517, 543
passing parameters 517, 543
remote 518, 544
returning result set 516, 543
schemas for 518, 544
stored procedures for Informix SQL command restrictions 669
string data
Aster 499
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 568
Greenplum 599
HAWQ 645
SAP HANA 842
Sybase IQ 888
Vertica 951
STRING data type
Hive (Hadoop) 616
Impala 659
string data types
Impala 659
STRINGDATES= LIBNAME option 218
STRUCT data type
Hive (Hadoop) 616
SUBMIT statement 979
CV2VIEW procedure 976
SUBPROTOCOL= connection option
Hadoop 608
SUBSET ACCESS procedure 957
SUBSET statement
ACCESS procedure 968
subsetting DBMS data 328
subsystem identifier (DB2) 560
subsystem name (DB2) 560
supported data types
DB2 under z/OS 568
Sybase 850
ACCESS procedure 858
autopartitioning scheme 856
bulk copy for loading tables 159
bulk loading 862
case sensitivity 865
character data 866
data conversions, ACCESS procedure 869
data conversions, DBLOAD procedure 870
data conversions, LIBNAME statement 867
data returned as SAS binary data, default format $HEX 871
data returned as SAS character data 871
data set options 854
data types 866
database links 339
date, time, and money data 866
DBLOAD procedure 859
DBSLICE= data set option 347
indexes 856
inserting TEXT data from SAS 871
LIBNAME statement 850
linking from default database to another database 142
locking 863
maximum simultaneous connections allowed 175
name and location of interfaces file 170
naming conventions 864
national language support 871
NULL values 867
numeric data types 866
packet size 180
Index

1044

T

table
load from an existing data set 253
table columns, numeric
specifying the default data type 178
table guid
preserving 186
table names
embedded spaces and special characters
199
fully qualified (Informix) 678
preserving 20, 191
qualifying 106
table objects 251
table properties, existing
Hive (Hadoop) 620
table properties, SAS
Hive (Hadoop) 619
table types
MySQL 703
TABLE_TYPE= data set option 411
TABLE_TYPE= LIBNAME option 220
TABLE=
ACCESS procedure 957
DBLOAD procedure 987
TABLE= option
DB2UTIL procedure 551
TABLE= statement, ACCESS procedure 968
TABLE= statement, DBLOAD procedure ODBC 748
tables 171
See also DB2 tables
See also DBMS tables
See also temporary tables
bulk copy for loading 159
catalog tables 448
creating with data sets 25
creating with DBMS data 24
data accessible during import 243
database or table space for creating 167
database or tablespace for creating 364
dimension tables 155, 354
duplicate rows 409
emptying with SQL DELETE statement 46
exception tables 281
fact tables 155, 183, 354, 391
location of 171, 367
original data visible during bulk loading 243
read time 53
reading from and inserting to same
Teradata table 918
redirecting log tables to alternate
database 173

passing joins to 862
passing SAS functions to 861
passing SQL functions to 861
reading multiple tables 862
SQL pass-through facility 855
supported features 88
temporary tables 858
update rules 864
user-defined data 867
Sybase IQ 874
autapartitioning scheme 881
bulk loading 884
data conversions 889
data set options 878
data types 887
date, time, and timestamp data 888
DBSLICE= data set option 347, 882
DBSLICEPARM= LIBNAME option 882
LIBNAME statement 874
locking 885
naming conventions 887
NULL values 888
nullable columns 882
numeric data 888
passing joins to 884
passing SAS functions to 883
special catalog queries 880
SQL pass-through facility 880
string data 888
supported features 89
WHERE clauses 882
SYBPW=
  PASSWORD= connection option for
  Sybase 851
synonyms 211, 219
SYNONYMS= LIBNAME option 219
SYSDBMSG macro variable 445
SYSDBRC macro variable 445, 560
SYSDISC data set name 259
SYSIN data set name 260
SYSMAP data set name 264
SYSPRINT data set name 264
SYSPRINT output 265
SYSREC data set
  name of 265
  number of cylinders 266
system catalogs
  DB2 under z/OS 581
  system options 445, 447
  DB2 under z/OS 559
  system-directed access 579
SYSTEM_TIMEFRAME= data set option 410
renaming segments where created temporary versus permanent tables, fact column names as partition key tables, Hive creating and deleting specifying HDFS location TABLESPACE=
    IN= data set option IN= LIBNAME option task table TBLSET
    SET= data set option TD_RACF environment variable
    Teradata under z/OS TDPIP= connection option
    Teradata TEMPORAL 412
temporal data DB2 under z/OS 573
temporal data for DB2 overview 573
temporal data types Teradata 934
    Temporal tables 319, 410
TEMPORAL_QUALIFIER= data set option TEMPORAL_QUALIFIER= LIBNAME option TEMPORAL= data set option TEMPORAL= data set option
    temporary files
directory for bulkloading temporary tables 143 acquisition error tables 369, 370
    DB2 under z/OS 542
    Informix 672
    MultiLoad 374
    Netezza 719
    ODBC 749
    OLE DB 769
    performance and 40
    prefix for names of 371
    pushing heterogeneous joins 41
    pushing updates 42
    restart table 372
    Sybase 858
    Teradata 904
    work table 373
    TENACITY= data set option TENACITY= data set option
    MultiLoad and TENACITY= LIBNAME option Teradata 892
See also TPT API
ANSI mode or Teradata mode autopartitioning scheme binary string data BL_DATAFILE= data set option buffers and transferring data to BYTEINT data type CHAR data type character string data checkpoint data data conversions data conversions, overhead limit for data returned as SAS binary data with default format $HEX data set options data types data types, temporal DATE data type date, time, and timestamp data DECIMAL data type encryption failed load process FastExport Utility FastExport Utility, logging sessions FastLoad FastLoad error limit FastLoad logging sessions FLOAT data type generating SQL for in-database processing of source data insert statements INTEGER data type LIBNAME statement load performance locking maximum number of sessions minimum number of sessions MultiLoad MultiLoad, logging sessions name of first error table name of restart log table name of second error table naming conventions NULL values number of sessions numeric data types ordering query results output buffer size overhead limit for data conversions packing statements passing joins to passing SAS functions to processing tips reading from and inserting to same table
partitioning WHERE clauses with SAS 917
Sybase 856
Sybase IQ 881
Teradata 914
trace information for 455
threaded Reads 53
autopartitioning and controlling number and scope of 151, 349, 451
data ordering and 60
data set options affecting 55
DB2 under z/OS 540
generating trace information for 56
ODBC 743
partitioning queries for 347
performance and 53, 58
SAP HANA 838
SAS/ACCESS interfaces and scope of 54
summary of 61
two-pass processing and 60
underlying technology of 54
when threaded Reads do not occur 61
throughput of SAS server 37
TIME data type
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 569
Greenplum 600
HAWQ 646
Netezza 724
Sybase 866
Sybase IQ 888
Teradata 920, 931
time-dimensional criteria for Teradata 221
TIME and TIMESTAMP 920
time-dimensional criteria for Teradata 412
time, timestamp values
Hive (Hadoop) 616
timeouts
default login timeout 174
for commands 323
for data source commands 117
queries 198, 399
timestamp 127, 326
TIMESTAMP data type
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 569
Greenplum 600
HAWQ 646
Netezza 724
Oracle 803
Sybase 866
Sybase IQ 888
Teradata 920, 931

redirecting log tables to alternate database 173
replacing SORT procedure with BY clause 921
restarting failed runs 427
sampling 922
SAS/ACCESS client 892
sending insert statements to 178
sharing objects with SAS 930
SMALLINT data type 931
SQL pass-through facility 901
supported features 90
table-level definitions 329
temporary tables 904
threaded reads with partitioning WHERE clauses 917
TIME and TIMESTAMP 920
TIME data type 931
time-dimension criteria for 221, 412
TIMESTAMP data type 931
tracing levels 430, 431
tracing output 432
utilities, retries for logging in 212, 222, 410, 415
VARCHAR data type 931
work table name 433
Teradata Parallel Transporter
See TPT API
Teradata server
interrupting SQL processes 226
Teradata Wallet 922
TERMCMD
DBCONTERM= LIBNAME option 135
termination command
executing before every disconnect 135
executing once 141
user-defined 135, 141, 475
TEXT data type
Greenplum 599
HAWQ 645
Sybase 866, 871
TEXTSIZE
DBMAX_TEXT= LIBNAME option 142
threaded applications 54
two-pass processing for 60
threaded reads
Aster 494
DB2 under UNIX and PC Hosts 513
Greenplum 592
HAWQ 638
Informix 671
number of connections to DBMS server for 326
Oracle 789

throughput of SAS server 37
TIME data type
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 569
Greenplum 600
HAWQ 646
Netezza 724
Sybase 866
Sybase IQ 888
Teradata 920, 931
time-dimensional criteria for Teradata 221
time-dimensional criteria for Teradata 412
time, timestamp values
Hive (Hadoop) 616
timeouts
default login timeout 174
for commands 323
for data source commands 117
queries 198, 399
timestamp 127, 326
TIMESTAMP data type
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 569
Greenplum 600
HAWQ 646
Netezza 724
Oracle 803
Sybase 866
Sybase IQ 888
Teradata 920, 931
TIMESTAMP WITH LOCAL TIME data type
Oracle 803
TIMESTAMP WITH TIME ZONE data type
Oracle 803
TINYINT data type
Hive (Hadoop) 616
Impala 659
Sybase IQ 888
TO_LIBREF= statement 979
CV2VIEW procedure 980
TO_VIEW= statement 979
CV2VIEW procedure 979
TPT API 905
See also FastLoad
See also MultiLoad
data set options 906
loading data 223, 416
multi-statement insert features and restrictions 909
multi-statement insert with TPT API data set options 909
reading data 161
setup 906
starting multi-statement insert 909
supported features and restrictions 905
TPT_APPL_PHASE= data set option 417
TPT_BUFFER_SIZE= data set option 418
TPT_CHECKPOINT_DATA= data set option 419
TPT_DATA_ENCRYPTION= data set option 421
TPT_ERROR_TABLE_1= data set option 422
TPT_ERROR_TABLE_2= data set option 423
TPT_LOG_TABLE= data set option 424
TPT_MAX_SESSIONS= data set option 425
TPT_MAX_SESSIONS= LIBNAME option 224
TPT_MIN_SESSIONS= data set option 426
TPT_MIN_SESSIONS= LIBNAME option 225
TPT_PACK= data set option 426
TPT_PACKMAXIMUM= data set option 427
TPT_RESTART= data set option 427
TPT_TRACE_LEVEL_INF= data set option 431
TPT_TRACE_LEVEL= data set option 430
TPT_TRACE_OUTPUT= data set option 432
TPT_WORK_TABLE= data set option 433
TPT= data set option 416
TPT= LIBNAME option 223, 906
TR_ENABLE_INTERRUPT= LIBNAME option 226
trace information
filename for 227
for debugging 227
for threaded reads 455
for threaded Reads 56
generating from DBMS engine 455
trace messages
location of 464
TRACE= LIBNAME option 227
TRACEFILE= LIBNAME option 227
tracking errors
acquisition error tables 369, 370
transactions control 32
TRANSCODE_FAIL= data set option 434
TRANSCODE_FAIL= LIBNAME option 228
TRAP151= data set option 434
triggers 28
TRUNCATE_BIGINT
environment variable 724
truncation
forcing during insert processing 333
names 15
two-pass processing
for threaded applications 60
TYPE
DBLOAD procedure 987
TYPE= statement 980
CV2VIEW procedure 980
U
UDL_FILE= connection option
OLE DB 758
UID=
USER= connection option for Microsoft SQL Server 682
USER= connection option for ODBC 732
USER= connection option for Vertica 940
UNION operator
passing to DBMS 48
UNIQUE
ACCESS procedure 957
UNIQUE statement
ACCESS procedure 969
UNIX
  See also DB2 under UNIX and PC Hosts
  ODBC on 729, 730
  unsupported characters replacing 19
  updatable cursors 125
  UPDATE
    ACCESS procedure 957
  Update operator 908
  Update privileges 560
  update rules
    MySQL 703
    SAP HANA 837
    Sybase 864
    Vertica 947
  UPDATE statement
    ACCESS procedure 969
    DB2UTIL procedure 552
    SQL procedure 482
    UPDATE_ISOLATION_LEVEL= data set option 436
    UPDATE_ISOLATION_LEVEL= LIBNAME option 228
    UPDATE_LOCK_TYPE= data set option 436
    UPDATE_LOCK_TYPE= LIBNAME option 229
    UPDATE_MODE_WAIT= data set option 437
    UPDATE_MODE_WAIT= LIBNAME option 230
    UPDATE_MULT_ROWS= LIBNAME option 231
    UPDATE_SQL= data set option 438
    UPDATE_SQL= LIBNAME option 231
    UPDATEBUFF= data set option 232, 438
    UPDATEBUFF= LIBNAME option 232
  updating
    access descriptors 969, 972
    committing immediately after submitting 107
    data 69
    DB2 tables 549, 553
    method for updating rows 438
    non-updatable columns 434
    pushing updates 42
    specifying number of rows 438
  upsert
    multiloading 439
    UPSERT_CONDITION multiloading 440
    UPSERT_CONDITION= data set option 440
    MultiLoad and 913
    upsert_where
    multiloading 441
    UPSERT_HERE= data set option
      MultiLoad and 913
    UPSERT= data set option 439, 441
      MultiLoad and 912
    USE_ODBC_CL= LIBNAME option 233
    USE_PROGRESS=
      USE_DATADIRECT= LIBNAME option 233
    user IDs 242
    qualifying table names with 106
    user name
      case sensitivity 193
      SFTP access 312
    user-defined data
      Sybase 867
    user-defined initialization command 133, 140, 475
    user-defined termination command 135, 141, 475
    USER= connection option
      Aster 487
      DB2 under UNIX and PC Hosts 505
      Greenplum 585
      Hadoop 606
      HAWQ 631
      Impala 650
      Informix 665
      Microsoft SQL Server 682
      MySQL 698
      Netezza 713
      ODBC 732
      OLE DB 756
      Oracle 780
      PostgreSQL 812
      SAP HANA 830
      Sybase 850
      Sybase IQ 875
      Teradata 893
      Vertica 940
    USER= database description statement
      Sybase 859
    USER= statement, DBLOAD procedure
      Microsoft SQL Server 690
      ODBC 747
    USING= connection option
      Informix 665
    UTILCONN_TRANSIENT= LIBNAME option 234
    utility connections 234
    utility spool files 212
VALIDVARNAME= system option 186, 469
naming behavior and 17
VALIDVARNAME=V7 argument
CONNECT statement 476
VARBINARY (BYTEA) data type
Greenplum 600
VARBINARY data type
Netezza 724
VARBYTE data type
Teradata 931
VARCHAR data type
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 568
Greenplum 599
HAWQ 645
Impala 659
Netezza 724
Sybase 866
Sybase IQ 888
Teradata 931
VARCHAR data type columns
adjusting default for 116
adjusting lengths for 102
specified with byte semantics 102
VARCHAR(n) data type
Hive (Hadoop) 616
VARCHAR2 column lengths 128
VARCHAR2 data type
Oracle 802
VARGRAPHIC data type
DB2 under UNIX and PC Hosts 527
DB2 under z/OS 568
variables
dropping before creating a table 987
generating names of 960
labels as DBMS column names 338
listing information about, before loading 987
macro variables 445
modifying names 964
names as DBMS column names 338
names based on column names 969
renaming 16
valid names during a SAS session 469
vendor
Hadoop distribution 159
Vertica 939
binary string data 951
data set options 943
data types 951
LIBNAME statement 940
locking 949
naming conventions 950
NULL values 952
numeric data 951
passing joins to 948
passing SAS functions to 947
SQL pass-through facility 945
string data 951
supported features 90
update and delete rules 947
view descriptors 970, 971
See also SAS/ACCESS views
converting a library of 983
converting an individual 980
creating 961, 972
dropping columns to make unselectable 962
listing columns in, with information 964
name, for converting 977
reading data with 68
resetting columns to default settings 966
selecting DBMS columns 967
selection criteria, adding or modifying 968
updating 969
VIEWDESC= option
PROC ACCESS statement 959
views 6
See also SQL views
access methods 4
data security 31
DATA step views 6
reading from 99
SAS data views 6
SAS/ACCESS views 7
SQL Server partitioned views for DBSLICE= 744
writing to 99
volumes
for extending data sets 270

WARN_BIGINT= LIBNAME option 234
warning: big integer 234
warnings 310
row warnings 316
Web server
de file location for segment host access 296
web tables
accessing dynamic data in 597, 643
WHERE clauses 49
Aster 494
DB2 under UNIX and PC Hosts 513
efficient versus inefficient 49
format of, with DBKEY= data set option 145
Greenplum 592
HAWQ 638
Informix 671
NULL values and format of 343
ODBC 743
partitioning queries for threaded Reads 347
passing functions to DBMS with 49
passing to DBMS 49
restricting autopartitioning 541
SAP HANA 838
Sybase IQ 882
threaded reads and partitioning WHERE clauses 917, 918

updating rows with no locking 180, 386
WHERE statement
DB2UTIL procedure 552
DBLOAD procedure 987
work table 373

X
XMS (Cross Memory Services) 578

Z
z/OS
See also DB2 under z/OS
Oracle bulk loading 799
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.