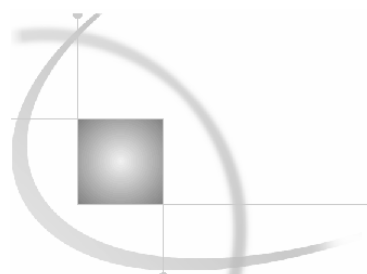


## ETL Performance Tuning Tips



# ETL Performance Tuning Tips

---

<b>Introduction</b> .....	1
<b>Overview</b> .....	1
<b>Important Concepts</b> .....	2
Jobs, Flows, and Transformations in SAS Data Integration Studio .....	2
Executing SAS Data Integration Studio Jobs .....	2
Identifying the Server That Executes a Job .....	3
Intermediate Files for SAS Data Integration Studio Jobs .....	4
Deleting Intermediate Files .....	4
<b>The Basics: Things to Consider</b> .....	5
Data Partitions Using Relational Databases or SAS Scalable Performance Data Server .....	5
Transformation and Column Optimizations .....	6
Extra Mappings .....	7
Tips for Sizing Input Data to an ETL Flow .....	7
Columns Input to the ETL Flow .....	7
Number of Bytes That Each Column Consumes .....	7
Rows Input to the ETL Flow .....	8
SAS Data Files Compression .....	8
Data Validation and Data Cleansing .....	9
Star Schemas and Lookups .....	9
Surrogate Keys .....	9
Remote versus Local Data .....	10
Joins, Views, and Physical Tables .....	10
Dormant Data .....	11
Simple Techniques for Writing ETL Flows .....	11
Display Job Status .....	11
Verify Transformation's Output .....	11
Limit Transformation's Input .....	11

Select a Load Technique .....	14
<b>SAS Data Integration Studio Process Flow Analysis .....</b>	<b>17</b>
Applying SAS Invocation Options to ETL Flows .....	17
SAS Invocation Options for SAS Data Integration Studio Jobs and Transformations .....	18
SAS Logs for Analyzing ETL Flows .....	18
Review SAS Logs during Job Execution .....	18
SAS Options for Analyzing Job Performance .....	19
Re-Direct Large SAS Logs to a File .....	20
View or Hide the Log in SAS Data Integration Studio .....	20
Debugging Transformations in an ETL Flow .....	21
Transformation Output Tables Used to Analyze ETL Flows .....	22
Viewing the Output File for a Transformation .....	22
SAS Options That Preserve Files Created during Batch Jobs .....	22
Re-Directing Output Files for a Transformation .....	23
List Data Transformation Added to the ETL Flow .....	24
User-Written Code Transformation Added to the ETL Flow .....	24
<b>Simple Tuning of ETL Flows Results in Immediate Gains .....</b>	<b>25</b>
Sorting Data .....	25
Tuning Sort Performance .....	26
Disk Space Consumption When Sorting .....	30
SAS SQL Joins .....	35
Optimizing Join Performance .....	35
Sort-Merge Joins .....	37
Index Joins .....	38
Hash Joins .....	41
Multi-Way Joins .....	42
Relational Database Considerations .....	43
Indexes on SAS Data Sets .....	45
WHERE Clauses Resolved in ETL Flows .....	45
Allowing Index Joins .....	45
Verifying Key Uniqueness .....	46
Index Used in SAS? .....	46
Encouraging Index Usage .....	47
Indexes for Sorting .....	47

<b>Environment Tuning</b> .....	<b>48</b>
Disk Space Maximum for Intermediate Files .....	48
Disk Space Usage for Intermediate Files .....	48
Deleting Transformation Output Tables in a Long ETL Flow .....	50
Hardware Considerations for ETL Flows .....	50
File Cache Management .....	51
I/O Performance Improvement .....	51
Direct I/O .....	52
Monitoring SAS Data Integration Studio Flows .....	53
<b>Advanced Topics</b> .....	<b>54</b>
Removing Non-Essential Indexes and Constraints During a Load .....	54
Optimizing Join Order and Performance .....	55
GROUPINTERNAL Option in PROC MEANS .....	55
Lookups in a Star Schema .....	55
<b>Hash-Object Lookup</b> .....	56
<b>Format-Based Lookup</b> .....	59
Performance Considerations with Lookup .....	61
Processing Level Performed by a Relational Database .....	61
UPCASE and LOWCASE Functions versus UPPER and LOWER Functions .....	62
Getting Counts of the Number of Rows in a Table .....	64
Row Counts in SAS Data Sets .....	64
Row Counts in Databases .....	66
Bulk Loading the Relational Database .....	66
SAS Date Functions .....	66
Views or Physical Tables for ETL Flows .....	68
Views or Physical Tables for SAS Data Integration Studio Flows .....	69
Loading Fact and Dimension Tables When Working with Dimensional Models .....	70
Overview of Dimensional Modeling .....	70
Transformations for Generating Keys and Loading Dimension Tables .....	70
Transformations for Loading Fact Tables and Matching Fact Tables to Dimension Tables .....	71
Fact and Dimension Tables: Order of Loading Is Important .....	71
SAS Scalable Performance Data Server Star Joins .....	71
Parallel ETL Processing .....	73
Software and Hardware Setups for Executing ETL Code in Parallel .....	74

Software Setup.....	74
Hardware Setup .....	74
System Throttles .....	75
File System Requirements for Grid Computing.....	76
How to Approach a Parallel ETL Effort.....	76
<b>Appendix 1: Customizing or Replacing Generated Code in SAS Data Integration Studio .....</b>	<b>78</b>
Modify Configuration Files or SAS Start Commands.....	78
MPRINT Option in SAS Data Integration Studio .....	79
Specify Options on the Code Generation Tab.....	79
Add SAS Code on the Pre- and Post-Processing Tab .....	79
SAS System Options for Transformations.....	80
Replace Generated Code for a Transformation with Customized Code.....	81
Add Your Own Code to User-Written Code Transformation.....	82
Add Your Own Transformation to the Process Library .....	83
<b>Appendix 2: A Sample Scenario for Using Parallel ETL Processing .....</b>	<b>84</b>
<b>Recommended Reading .....</b>	<b>88</b>
<b>Glossary.....</b>	<b>89</b>

---

## Introduction

A team of ETL performance experts at SAS Institute reviewed ETL flows for several SAS®9 solutions with the goal of improving the performance and scalability of the solutions. The recommendations that are presented here are based on team observations and performance testing. A review of concepts that are important to understanding the content of this paper is given at the beginning of the paper, and a Glossary is provided at the end of the paper.

---

## Overview

This paper provides important ETL (extract, transform, and load) performance, tuning, and capacity information for SAS®9 and SAS Data Integration Studio. ETL concepts are introduced, and ETL performance topics are discussed in depth. Examples are given, as needed. Topics include how to analyze and debug flows, gain efficiency quickly, set up the system environment, and, when necessary, customize advanced performance techniques. Many best practice recommendations are presented that will help you to customize and enhance the performance of new and existing ETL processes.

This paper is written for developers, administrators, and project leaders who are responsible for data integration and who have an understanding of the basic principles of data warehouse ETL processes. In addition, business sponsors, project managers, and IT support staff might find the material helpful when they provide guidelines for designing ETL processes that scale up to meet your requirements across your enterprise.

Many examples in this paper require a basic understanding of data warehousing and relational database concepts such as tables, rows, keys, joins, and views. It is helpful (but not required) for you to understand the SAS language. It is also helpful if you have knowledge of the SAS®9 architecture and the role that metadata plays in the architecture. For more information about the SAS®9 architecture read the *SAS Intelligence Platform: Overview* available at [support.sas.com/documentation/configuration/biov.pdf](http://support.sas.com/documentation/configuration/biov.pdf).

The recommendations that are presented in this paper are applicable:

- to SAS 9.1.3.
- specifically to ETL flows that use Base SAS data sets as the data store. As appropriate, commentary and best practices are applied to discussions about flows that use data from the SAS Scalable Performance Data Server (SAS SPD Server) and relational databases.
- across hardware platforms and operating environments except where noted.
- to code that is generated by SAS Data Integration Studio and to custom code that is developed outside SAS Data Integration Studio.

---

## Important Concepts

---

### Jobs, Flows, and Transformations in SAS Data Integration Studio

In SAS Data Integration Studio, a **job** is a collection of SAS tasks that specify processes that create output. Each job generates or retrieves SAS code that reads **sources** and creates **targets** in physical storage. To generate code for a job, you must create a **process flow diagram** that defines the sequence of each source, target, and process in a job. Each process in an **ETL flow** is specified by a task called a transformation. A **transformation** specifies how to extract data, transform data, or load data into data stores. Each transformation generates or retrieves SAS code. You can specify user-written code for any transformation in an ETL flow.

Figure 1 shows an ETL flow for a job that reads data from a source table, sorts the data, and then writes the sorted data to a target table.

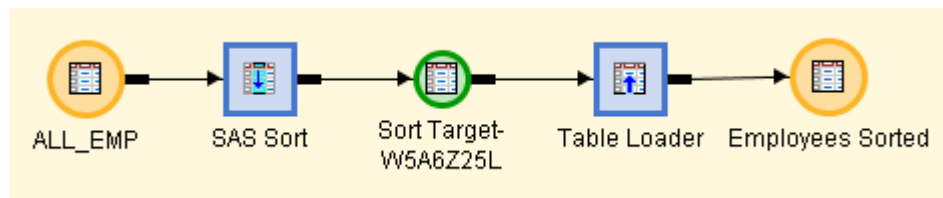


Figure 1. SAS Data Integration Studio Process Flow Diagram for a Job That Sorts Data

The SAS data set ALL\_EMP specifies metadata for the source table. The SAS Sort transformation sorts the input data to the temporary output table Sort Target-W5A6Z25L. The Table Loader transformation reads the output from the previous task and loads this data into a target table. The SAS data set Employees Sorted is the target table. SAS Data Integration Studio uses metadata to generate SAS code that reads the data set ALL\_EMP, sorts this information, writes the sorted information to a temporary output table, then writes it to the Employees Sorted table.

---

### Executing SAS Data Integration Studio Jobs

You can execute a SAS Data Integration Studio job by using one of the following options:

- the `Submit Job` option to submit the job for interactive execution on a SAS Workspace Server.
- the `Deploy for Scheduling` option to generate code for the job and save it to a file; then, you can execute the job in batch mode.
- the `Stored Process` option to generate a stored process for the job and save it to a file; then, you can execute the job via a stored process server.
- the `Web Service` option to generate a Web service for the job and save it to a hosting server; then, you can execute the job via a Web server.
- the `SAVE AS` option to save the job to a file that can be executed later.

---

## Identifying the Server That Executes a Job

In Business Intelligence Architecture applications in SAS®9 such as SAS Data Integration Studio, a **SAS Application Server** represents an existing server environment that contains servers, libraries, schemas, directories, and other resources. Usually, an administrator defines this environment and tells the SAS Data Integration Studio user which server to select as the default (usually, the default is SASMAIN).

Behind-the-scenes, when you submit a SAS Data Integration Studio job for execution, it is submitted to a SAS Workspace **Server component** of the relevant SAS Application Server. The relevant SAS Application Server is one of the following:

- the default server that is specified on the **SAS Server** tab in the SAS Data Integration Studio Options window
- the SAS Application Server that a job is deployed to when the `Deploy for Scheduling` option is used.

It is important to know which SAS Workspace Server or which servers will execute a job in order to perform the following tasks:

- store data where it can be accessed efficiently by the transformations in a SAS Data Integration Studio job, as described in the section "Remote versus Local Data".
- locate the SAS WORK library where the job's intermediate files are stored by default
- specify the SAS options that you want to apply to all jobs that are executed on a given server, as described in the section "SAS Invocation Options for SAS Data Integration Studio Jobs and Transformations".

To identify the SAS Workspace Server or the servers that will execute a SAS Data Integration Studio job, use **SAS Management Console** to examine the metadata for the relevant SAS Application Server.

---

## Intermediate Files for SAS Data Integration Studio Jobs

Transformations in a SAS Data Integration Studio job can produce three types of **intermediate files**:

- **procedure utility files** are created by the SORT and SUMMARY procedures if these procedures are used in the transformation
- **transformation temporary files** are created by the transformation as it is working
- **transformation output tables** are created by the transformation when it produces its result; the output for a transformation becomes the input to the next transformation in the flow.

For example, suppose you execute the job with the ETL flow that is shown in Figure 1. When the SAS Sort is finished, it creates an output table. The default name for the output table is a two-level name that contains the libref WORK and a generated member name, for example, work.W5A6Z25L. This output table becomes the input to the next task in the ETL flow.

By default, procedure utility files, transformation temporary files, and transformation output tables are created in the WORK library. You can use the `-WORK` invocation option to force all intermediate files to a specified location. You can use the `-UTILLOC` invocation option to force only utility files to a separate location. (There are more details about this topic throughout this paper.)

Knowledge of intermediate files helps you to:

- view or analyze the output tables for a transformation and verify that the output is correct, as described in the section "Transformation Output Tables Used to Analyze ETL Flows".
- estimate the disk space that is needed for intermediate files, as described in the section "Disk Space Maximum for Intermediate Files".

---

## Deleting Intermediate Files

Utility files are deleted by the SAS procedure that created them. Transformation temporary files are deleted by the transformation that created them.

When a SAS Data Integration Studio job is executed in batch, transformation output tables are deleted when the ETL flow ends or the current server session ends.

When a job is executed interactively in SAS Data Integration Studio, transformation output tables are retained until the Process Designer window is closed or the current server session is ended in some other way (for example, by selecting **Process ► Kill** from the menu).

Transformation output tables can be used to debug the transformations in a job, as described in the section "Transformation Output Tables Used to Analyze ETL Flows". However, as long as you keep the job open in the Process Designer window, the output tables remain in the WORK library on the SAS Workspace Server that executed the job. If this is not what you want, you can manually delete the output tables, or you can close the Process Designer window and open it again, which will delete all intermediate files.

---

## The Basics: Things to Consider

Building efficient processes to extract data from **operational systems**, transforming the data into the appropriate data structures that form the data warehouse, and loading the data into those structures is critical to the success of the data warehouse. Efficiency becomes more important as data volumes and complexity increase. This section provides simple recommendations for creating ETL processes. Specific SAS Data Integration Studio optimizations and more general considerations are discussed. These techniques help ensure the success and longevity of the ETL project and the overall success of the data warehouse.

---

### Data Partitions Using Relational Databases or SAS Scalable Performance Data Server

The following advantages result from partitioning data:

- Updates--partitioning enables you to manage smaller tables that do not have to be uploaded as frequently.
- Performance--partitioning enables you to spread your data across your I/O subsystem (more disks) to increase your throughput capability.

Consider partitioning the source data and the target data on a boundary that best matches the needs of the consumers of the data warehouse. For example, if users will most often query information from the warehouse based on monthly data, it would be best to partition the data by month. Here are some other factors to consider when partitioning data:

- Select a natural boundary such as the Time boundary, which can minimize the size of the files that are being imported. Using weekly or monthly extracts of the data can considerably reduce the amount of data that has to be loaded or extracted from an operation source system. This can speed-up the delivery of the data to the transformations and minimize load times. Spreading these smaller partitions across your I/O subsystem helps parallel processes access the data more quickly. (Read more about parallel processing in the section "Parallel ETL Processing".)
- Select boundaries, by using WHERE clauses or key-based partitions, that describe the data. For example, partition the data by State, County, Social Security Number, or some other value that describes the data.

In addition, when you are partitioning, try to distribute the data as evenly as possible across the partitions so that one transformation doesn't have to read or load the bulk of the data at one time. If one partition is significantly larger than another partition, the benefit of the partitioning scheme is reduced.

---

## Transformation and Column Optimizations

For efficiency, try to minimize the number of times that the data is read and written to disk. Where possible, consider using views to access the data directly. Many transformations in SAS Data Integration Studio enable you to set the output to be a view instead of a physical table, as described in the section "Views or Physical Tables for SAS Data Integration Studio Flows". Using views eliminates extra Reads and Writes of the data, which use additional I/O cycles and take up disk space.

As the data is coming in, consider dropping any columns that are not required for subsequent transformations in the flow. If you drop columns and make aggregations early in the ETL flow instead of later, then extraneous detail data is not carried over to all the transformations in the flow. The goal is to create, early in an ETL flow, the structure that matches the final target table structure as closely as possible, so that extra data is not carried over.

To drop columns in the output table for a SAS Data Integration Studio transformation, click the **Mapping** tab and remove the extra columns from the **Target table** area on the tab. Use **derived mappings** to create expressions to map several columns together. You can also turn off automatic mapping for a transformation by right-clicking the transformation in the ETL flow, then deselecting the **Automap** option on the pop-up menu. Then, you can build your own transformation output table columns to match, as closely as possible, your final target table, and map columns as necessary.

When you map columns, consider the level of detail that is being retained. Ask yourself these questions:

- Is the data being processed at the correct level of detail?
- Can the data be aggregated?

Aggregations and summarizations eliminate redundant information and reduce the number of records that have to be retained, processed, and loaded into the data warehouse.

Also, because data volumes multiply quickly, you must ensure that the size of the variables that are being retained in the data warehouse is appropriate to the data length. When you do this, consider the current and the future uses of the data. Answering the following questions will help you determine the correct size variables for the data.

- Are the keys the right length for the current data?
- Will the keys accommodate future growth?
- Are the data sizes on other variables correct?
- Do the data sizes need to be increased or decreased?

---

## Extra Mappings

As data is passed from step-to-step in an ETL flow, columns can be added or modified. For example, column names, lengths, or formats might be added or changed. In SAS Data Integration Studio, these modifications to a table (via the transformation's **Mapping** tab) often result in generating an intermediate SQL view step. In many situations, that intermediate step adds processing time. Try to avoid generating more of these steps than is necessary.

Instead of performing column modifications or additions throughout many transformations in an ETL flow, re-work your flow so that these activities are consolidated in fewer transformations. Avoid using unnecessary aliases. If the mapping between columns is one-to-one, keep the same column names. Avoid multiple mappings of the same column, for example, don't convert a column from a numeric to a character value in one transformation, and then convert that column from a character to a numeric value in another transformation. For aggregation steps, re-name columns in the aggregation transformations instead of in subsequent transformations.

In addition to consolidating the intermediate SQL view steps, you might observe that the logic that is performed by one of these views can be performed by using different code in the preceding or the following generated steps. If necessary, you can incorporate user-written code into an ETL flow using one of the methods that are described in "Appendix 1: Customizing or Replacing Generated Code in SAS Data Integration Studio".

---

## Tips for Sizing Input Data to an ETL Flow

To size input data to an ETL flow, examine the following factors:

- number of columns that will be input to the ETL flow
- number of bytes that each column consumes
- number of rows that will be input to the ETL flow

For details about this topic, see the section "Sizing Input Data".

## Columns Input to the ETL Flow

You can subset the number of columns that are input to the ETL flow in order to create a "thinner" table by using, for example, a DROP or a KEEP clause or SQL that selects only some rows. Eliminated columns do not appear in the ETL flow.

## Number of Bytes That Each Column Consumes

The width of a SAS character column is represented by its format. For example, a character column that has the format \$30 holds 30 characters. In the usual condition, that is, SAS running in single-byte character mode, this column consumes 30 bytes. (If you run SAS in Unicode or in another multi-byte character mode, the character column width doubles, as a minimum.) With the exception of short numerics, other columns (such as numerics, dates, datetimes, and so on) consume 8 bytes per column. If input data is from a relational database, columns are translated to a SAS format. CHARACTER(30) and VARCHAR(30) columns are translated to \$30, which consumes 30 bytes. Other relational database columns translate to SAS types that consume 8 bytes. The CONTENTS procedure displays space consumption for each column. The following PROC CONTENTS statements and partial output

are for a relational database table 'input' that has four columns: name CHAR(30), age INTEGER, enrollment\_time TIMESTAMP, and description VARCHAR(64). The number of bytes per column appear under the column heading "Len".

```
proc contents varnum data=rdbms.input;
run;
```

Variables in Creation Order

#	Variable	Type	Len	Format	Informat	Label
1	name	Char	30	\$30.	\$30.	name
2	age	Num	8	11.	11.	age
3	enrollment_time	Num	8	DATETIME26.6	DATETIME26.6	enrollment_time
4	description	Char	64	\$64.	\$64.	description

### Rows Input to the ETL Flow

You can subset the number of rows that are input to an ETL flow in order to create a "shorter" table by using WHERE-clause filtering or other means, for example, the OBS and FIRSTOBS options. Eliminated rows do not appear in the ETL flow. Determining the initial count of input rows, before subsetting, is straightforward and efficient for SAS data. PROC CONTENTS lists the total row count as Observations. However, for relational database data, PROC CONTENTS does not indicate row count. You can determine row count for a relational database table by using SELECT COUNT(\*) in SQL, which is sometimes an expensive relational database operation. It is also expensive to pre-determine row count when your input is a SAS SQL view that translates to a join or other complex logic.

---

### SAS Data Files Compression

When working with "large" SAS data files, SAS users are often confronted with the problem of how to reduce the amount of disk space that's required to store the file without deleting important columns or rows. In such cases, use the COMPRESS=YES data set option. Most of the time, the COMPRESS=data set option reduces the size of your SAS data file but, if there are not a lot of repeating blanks, it might increase the size of the data file. There is also a cost of increased CPU utilization when executing DATA or procedure steps on the compressed data set, because the records in the file must be uncompressed before SAS can use them.

When compression is specified, depending on the characteristics of each observation in the data set, SAS removes repeating blanks and adds a "tag" to each observation that contains the information that SAS needs to uncompress each row when it is used. The question is whether the overhead of using compression is larger or smaller than the amount of storage space that is saved. SAS displays the results of applying the data set compression algorithm in the SAS log.

It is recommended that you specify the data set option COMPRESS=YES on a file-by-file basis and not use the SAS system global option COMPRESS=YES. The SAS system global option compresses every file (including very small temporary files) and the overhead of compression might degrade the performance of your SAS job.

For details about using compression with SAS, see the following paper, which gives the pros and cons for using SAS compression with your SAS data files: "Indexing and Compressing SAS® Data Sets: How, Why, and Why Not" at [www2.sas.com/proceedings/sugi28/003-28.pdf](http://www2.sas.com/proceedings/sugi28/003-28.pdf).

---

## Data Validation and Data Cleansing

To reduce the volume and increase the accuracy of the data that is sent through the ETL flow, validate and clean your data. If you need to, clean and delete duplicates in the incoming data early in the ETL flow so that extraneous data that might cause errors in the flow later in the process is eliminated.

You can clean the data by using the SAS Sort transformation with the NODUPKEY option or use the Data Validation transformation. In a single pass of the data, the Data Validation transformation detects missing-values and invalid values. It is important to eliminate extra passes over the data, so you should try to program all validations in a single transformation. The Data Validation transformation also eliminates duplicates and provides error-condition handling.



**Note:** You can also use the DataFlux (a SAS company, [www.dataflux.com](http://www.dataflux.com)) Data Quality transformations, apply Lookup Standardization, and create Match Code for **data cleansing**.

---

## Star Schemas and Lookups

When you are building ETL flows that require lookups (such as **fact table** loads and data validation), consider using the Lookup transformation that is available in SAS Data Integration Studio. The Lookup transformation is built by using a fast, in-memory lookup technique that is known as DATA step hashing. This transformation allows multi-column keys, has useful error-handling techniques, such as control over missing value handling, and the ability to set limits on errors. For details about lookups, see the section "Lookups in a Star Schema".

When you are working with **star schemas**, you can use the Slowly Changing Dimensions transformation. This transformation efficiently detects changed data and is optimized for performance. Several change-detecting techniques based on date, current indicator, and version number are supported. For details about the Slowly Changing Dimensions transformation, see the section "Loading Fact and Dimension Tables When Working with Dimensional Models".

---

## Surrogate Keys

Another technique to consider when you are building the data warehouse is to use incrementing integer **surrogate keys** as the main key technique in your data structures. **Surrogate keys** are values that are assigned sequentially, as needed, to populate a **dimension**. They are very useful because they can shield users from changes in the incoming data that might invalidate the data in a warehouse (and require re-design and re-loading). In this case, the surrogate key remains valid, but an operational key would not.

The Slowly Changing Dimensions transformation includes a surrogate key generator. You can also plug in your own methodology that matches your business environment to generate the keys and point the transformation to it. In addition, there is a Surrogate Key Generator transformation that builds incrementing integer surrogate keys.

Avoid character-based surrogate keys. In general, functions that are based on integer keys are more efficient because they avoid the need for subsetting or string partitioning that might be required for character-based keys. Also, numerics are smaller in size than character strings, thereby reducing the amount of storage that is required in the warehouse.

For more information about surrogate keys and the Slowly Changing Dimensions transformation, the section "Loading Fact and Dimension Tables When Working with Dimensional Models".

---

## Remote versus Local Data

Remote data has to be copied locally because it is not accessible by the relevant components in the default SAS Application Server at the time that the code was generated. SAS uses SAS/CONNECT and the UPLOAD and DOWNLOAD procedures to move data. It can take longer to access remote data than local data, especially when you access large data sets.

For example, the following data is considered local in a SAS Data Integration Studio job:

- data that can be accessed as if it were on the same computer(s) as the SAS Workspace Server component(s) of the default SAS Application Server
- data that is accessed with a SAS/ACCESS engine (used by the default SAS Application Server).

The following data is considered remote in a SAS Data Integration Studio job:

- data that cannot be accessed as if it were on the same computer(s) as the SAS Workspace Server
- data that exists in a different operating environment from the SAS Workspace Server component(s) of the default SAS Application Server (such as mainframe data that is accessed by servers running under Microsoft Windows).

---

## Joins, Views, and Physical Tables

Many ETL flows require table joins that can be resource-intensive. For details about optimizing joins, see the section "SAS SQL Joins".

In general, each task in an ETL flow creates an output table that becomes the input of the next task in the flow. Consider which format would be best for transferring data between s in the flow. For a discussion of the relative merits of views and physical tables in an ETL flow, see the section "Views or Physical Tables for ETL Flows".

---

## Dormant Data

Dormant data is data that exists in the warehouse but does not contribute to current usage patterns. As a data warehouse evolves over time, data ages. Data that does not contribute to queries is taking valuable space and time. By removing dormant data, user-access times can be improved. Consider archiving aged-out data. Dimensional data can be aged-out based on whether records are current. Other techniques include partitioning the data and storing it based on time, such as monthly or yearly values.

---

## Simple Techniques for Writing ETL Flows

When you build ETL flows, begin with simple tasks and build up to complex tasks instead of beginning with complex tasks. For example, build multiple, individual jobs and validate each job instead of building large, complex jobs. This will ensure that the simpler logic produces the expected results.

Also, consider subsetting incoming data or setting a pre-process option to limit the number of rows that are initially processed. This enables you to fix job errors and validate results before applying the processes to large volumes of data or to complex tasks. For more information about adding a pre-process to a SAS Data Integration Studio job, see the section "Add SAS Code on the Pre- and Post-Processing Tab".

## Display Job Status

After you submit one or more jobs for execution, display the Job Status Manager window to view the name, status, starting time, ending time, and application server that will be used for all jobs that are submitted in the current session. From the SAS Data Integration Studio desktop, select **Tools ► Job Status Manager** to open the Job Status Manager window.

## Verify Transformation's Output

If a job is not producing the expected output or if you suspect that something is wrong with a specific transformation, you can view the output tables for the transformations in the job in order to verify that each transformation is creating the expected output. See the section "Transformation Output Tables Used to Analyze ETL Flows".

## Limit Transformation's Input

When you are debugging and working with large data files, you might find it useful to decrease some or all the data that is flowing into a specific task. Here are some suggestions for doing this:

- Use the `OBS=` data set option on input tables of DATA steps and procedures.

In SAS Data Integration Studio, you can do this with a simple edit in the source editor.

In version 2 of the SQL Join transformation, to limit observations, specify the following options: "Last Row to Process" and "First Row to Process" (optional). In a multi-table join process, you can specify these settings individually for each table that participates in the join.

In the following example, the query will pull only 1000 rows from the fact table, but it will find matches from the complete **dimension table**.

```
proc sql;
  create table finalTable as
  select a.*, b.dimVar1
  from factTable(obs=1000) as a, dimTable as b
  where a.id=b.id;
quit;
```

In the example shown in Figure 2, the query that is shown in the bottom-right panel of the window will pull 10 rows from the PRODUCTS source table, beginning at row 10.

The screenshot shows the SAS SQL Properties window with the Designer tab selected. The 'PRODUCTS Properties' table is visible, showing 'Last Row to Process' set to 20 and 'First Row to Process' set to 10. The SQL Clauses panel shows 'Where', 'Group by', 'Having', and 'Order by' options. The main window displays a diagram of an inner join between 'PRODUCTS - PRODUCTS' and 'MANUFACTURERS - MANUFACTURERS'. The bottom-right panel shows the following SQL code:

```
1 proc datasets lib = BEYOND nolist nowarn memtype =
2   delete SAMPLE1;
3   quit;
4
5 proc sql
6   create table BEYOND.SAMPLE1 as
7   select
8     PRODUCTS.PRODNAME length = 25,
9     PRODUCTS.PRODCOST length = 8
10      format = DOLLAR9.2,
11     MANUFACTURERS.MANUNUM length = 8,
12     MANUFACTURERS.MANUNAME length = 22
13   from
14     BEYOND.PRODUCTS
15   (
16     obs = 20
17     firstobs = 10
```

Figure 2. Subsetting Rows in Version 2 of SQL Join Transformation

- Specify the option on the transformation to pull in only a specific number of input observations from all tables that are used in the query, as shown in Figure 3.

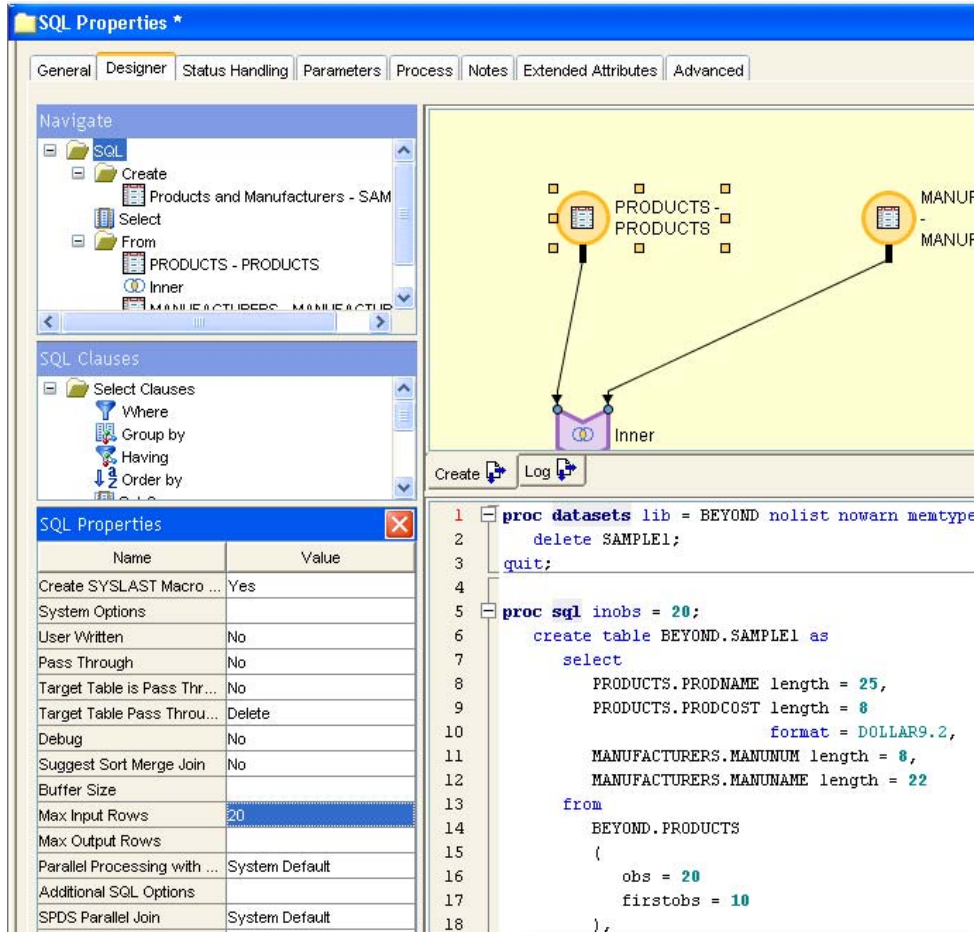


Figure 3. Setting the Number of Input Rows in Version 2 of SQL Join Transformation

- Insert the OBS= system option into the desired part of your process flow.

In SAS Data Integration Studio, you can do this by adding the option for a temporary amount of time on a transformation's **Options** tab, or by adding the following **OPTIONS** statement on the **Pre and Post Processing** tab in the job's Property window.

```
options obs=<number>;
```

Here are some important facts to consider when you use the OBS= system option.

- All inputs to all subsequent tasks will be limited to the specified number until you re-set the OBS= system option.
- Setting the value for the OBS= system option too low, prior to a join or a merge, can result in few or no matching records, depending on the data.
- In the SAS Data Integration Studio Process Editor, the OBS= system option stays in effect for all runs of the job until it is re-set or the Process Designer window is closed. The syntax for re-setting the option is:

```
options obs=MAX;
```



**Note:** Removing the preceding line of code from the Process Editor does not re-set the OBS= system option. You must re-set it by using the preceding OPTIONS statement or by closing the Process Designer window.

## Select a Load Technique

An important step in an ETL process usually involves loading data into a permanent physical table that is structured to match your data model. As the designer or builder of an ETL process flow, you must identify the type of load that your process requires in order to: append all source data to any previously loaded data, replace all previously loaded data with the source data, or use the source data to update and add to the previously loaded data based on specific key column(s). When you know the type of load that is required, you can select the techniques and options that are available that will maximize the step's performance.

In SAS Data Integration Studio, you use the Table Loader transformation to perform any of the three load types (or "Load style" as they are labeled in Figure 4). The transformation generates the code that is required in order to load SAS data sets, database tables, and other types of tables, such as an Excel spreadsheet. When loading tables, the transformation maintains indexes and constraints on the table that is being loaded.

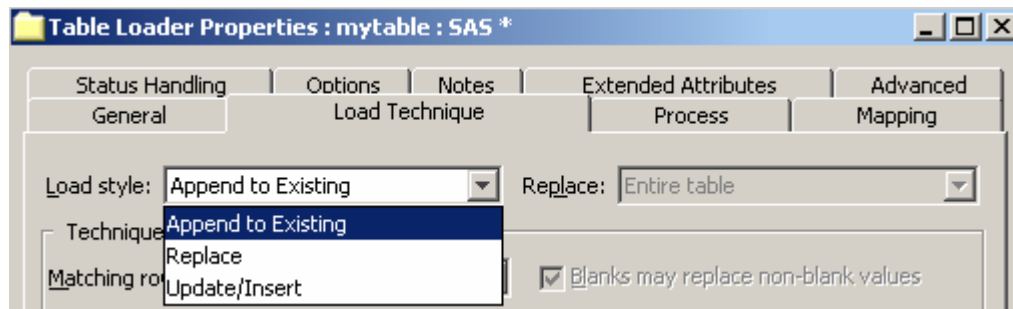


Figure 4. Load Styles on the Table Loader Transformation Load Technique Tab

After you have selected the Load style, you can choose from a number of load techniques and options. Based on the Load style that you select and the type of table that is being loaded, the choice of techniques and options will vary. The Table Loader transformation generates code to perform a combination of the following tasks:

- Remove all rows
- Add new rows
- Match rows
- Update rows

The following sections describe the SAS code alternatives for each task, and provide tips for selecting the load technique (or techniques) that will perform best.

### Remove All Rows

This task is associated with the **Replace** Load style. Based on the type of target table that is being loaded, two or three selections are listed in the "Replace" combo-box:

- Replace entire table – uses PROC DATASETS to delete the target table
- Replace all rows using truncate - uses PROC SQL with TRUNCATE to remove all rows (only available for some databases)
- Replace all rows using delete - uses PROC SQL with DELETE \* to remove all rows

When you select "Replace entire table", the table is removed and disk space is freed. Then the table is re-created with 0 rows. Consider this option unless your security requirements restrict table deletion permissions (a restriction that is commonly imposed by a database administrator on database tables). Also, avoid this method if the table has any indexes or constraints that SAS Data Integration Studio cannot re-create from metadata (for example, check constraints).

If available, consider using "Replace all rows using truncate". Either of the "Remove all rows..." selections enable you to keep all indexes and constraints intact during the load. By design, using TRUNCATE is the quickest way to remove all rows. The "DELETE \*" syntax also removes all rows; however, based on the database and table settings, this choice can incur overhead that will degrade performance. Consult your database administrator or the database documentation for a comparison of the two techniques.



**Caution:** When "DELETE \*" is used repeatedly to clear a SAS table, the size of that table should be monitored over time. "DELETE \*" only performs logical deletes for SAS tables, therefore, a table's physical size will grow and the increased size can negatively affect performance.

### Add New Rows

For this task, the Table Loader transformation provides two techniques for all three Load styles: PROC APPEND with the FORCE option and PROC SQL with the INSERT statement. The two techniques handle discrepancies between source and target table structures, differently. For details about PROC APPEND and PROC SQL, see the SAS Help documentation.

PROC APPEND with the FORCE option is the default. If the source is a large table and the target is in a database that supports bulk load, PROC APPEND can take advantage of the bulk-load feature. Consider bulk loading the data into database tables, by using the optimized SAS/ACCESS engine bulk loaders. (It is recommended that you use native SAS/ACCESS engine libraries instead of ODBC libraries or OLEDB libraries for relational database data. SAS/ACCESS engines have native access to the databases and have superior bulk-loading capabilities.) For more information about bulk loading, see the section "Bulk Loading the Relational Database".

PROC SQL with the INSERT statement performs well when the source table is small. This is because you don't incur the overhead that is necessary to set up bulk loading. PROC SQL with INSERT adds one row at a time to the database.

### Match Rows and Update Rows

The Table Loader transformation provides three techniques for matching and updating rows in a table. All these techniques are associated with the Update/Insert Load style.

- DATA step with the MODIFY BY
- DATA step with the MODIFY KEY=
- PROC SQL with the WHERE and SET statements

For each of these techniques, you must select a column (or columns) or an index for matching. All three techniques will update matching rows in the target table. The options `MODIFY BY` and `MODIFY KEY=` have the added benefit of being able to take unmatched records and add them to the target table during the same pass through the source table.

Of these three choices, the DATA step with `MODIFY KEY=` often out-performs the other update methods in tests conducted on loading SAS tables. An index is required. `MODIFY KEY=` can also perform adequately for database tables when indexes are used.

When PROC SQL with the WHERE or SET statement is used, performance varies. Neither of these statements in PROC SQL requires data to be indexed or sorted, but indexing on the key column(s) can greatly improve performance. Both of these statements use WHERE processing to match each row of the source table with a row in the target table.

The update technique that you choose depends on the percentage of rows being updated. If the majority of target records are being updated, the DATA step with MERGE (or UPDATE) might perform better than the DATA step with `MODIFY BY` or `MODIFY KEY=` or PROC SQL because MERGE makes full use of record buffers. Performance results can be hardware and operating-environment dependent, so you should consider testing more than one technique.



**Note:** Though the general Table Loader transformation does not offer the DATA step with MERGE as a load technique, you can revise the code for the MODIFY BY technique to do a merge and save that as user-written code for the transformation. (See "Appendix 1: Customizing or Replacing Generated Code in SAS Data Integration Studio".)



**Tip:** You should also take into consideration the number of indexes and the number of column constraints that exist. Temporarily removing these during the load can significantly improve performance. See the advanced topic "Removing Non-Essential Indexes and Constraints During a Load".

---

## SAS Data Integration Studio Process Flow Analysis

Occasionally, an ETL flow might run longer than you expect, or the data that is produced might not be what you anticipate (either too many records or too few). In such cases, it is important to understand how an ETL flow works, so that you can correct errors in the flow or improve its performance.

A first step in analyzing ETL flows is being able to access information from SAS that explains what happened during the run. If there were errors, you need to understand what happened before the errors occurred. If you're having performance issues, the logs will explain where time is being spent. Finally, if you know which SAS options are set and how they are set, this can help you find out what is going on in your ETL flows.

The next step in analyzing ETL flows is interpreting the information that you obtain. This section discusses how to use the SAS log to gather important information, how to analyze this information, how to determine option settings, how to get the return code from a job, and how to maintain the intermediate files after the ETL flow is completed so that you can review what is being created.

---

### Applying SAS Invocation Options to ETL Flows

Most SAS options can be specified in several locations, such as in configuration files, at invocation, and in SAS source code as global, libname, or data set options.

Invocation options override options specified in the configuration file, with the exception of SAS restricted options that are supported on the UNIX and z/OS operating environments. Restricted options are defined by SAS Administrators and cannot be overridden. The options that are suggested in this paper are, usually, not restricted by SAS Administrators. To see what configuration files are read at SAS startup, including any files that contain restricted options, submit the following code in your SAS session or SAS program:

```
proc options option=config value;  
  
run;
```

To see any restricted options that are supported on the UNIX or z/OS operating environments, submit the following code:

```
proc options restrict;  
  
run;
```

For further information about SAS configuration files and SAS invocation options, see the topics about "configuration" and "invoking SAS" in your SAS online documentation.

## SAS Invocation Options for SAS Data Integration Studio Jobs and Transformations

When you submit a SAS Data Integration Studio job for execution, it is submitted to a SAS Workspace Server component of the relevant SAS Application Server. The relevant SAS Application Server is one of the following:

- the default server that is specified on the **SAS Server** tab in the Options window
- the SAS Application Server to which a job is deployed with the `Deploy for Scheduling` option.

To set SAS invocation options for all SAS Data Integration Studio jobs that are executed by a particular SAS server, specify the options in the configuration files for the relevant SAS Workspace Servers, batch or scheduling servers, and grid servers. (Do not set these options on SAS Metadata Servers or SAS Stored Process Servers.)

You can set SAS system options for a particular job on the **Pre and Post Process** tab in the Properties window for a job. For details about adding a pre-process to a SAS Data Integration Studio job, see the section "Add SAS Code on the Pre- and Post-Processing Tab".

The Property window for transformations in a job has an **Options** tab with a **System Options** field. Use the **System Options** field to specify options for a particular transformation in a job's ETL flow.

---

## SAS Logs for Analyzing ETL Flows

The errors, warnings, and notes in a SAS log provide a wealth of information about ETL flows. However, large SAS logs can decrease performance, so the costs and benefits of large SAS logs should be evaluated. For example, you might not want to create large SAS logs by default, in a production environment.

### Review SAS Logs during Job Execution

The SAS logs from your ETL flows are an excellent resource to help you understand what is happening as the flows execute. For example, when you look at the run times shown in the log, compare the real-time values to the CPU time (user CPU plus system CPU). If there is a difference of 20-25%, then you should look at the hardware to see if there is a computer resource bottleneck. For more information about how the computer resources are being used by the SAS process, see "Solving SAS Performance Problems: Employing Host Based Tools" at [support.sas.com/rnd/scalability/papers/TonySUGI31\\_20060403.pdf](https://support.sas.com/rnd/scalability/papers/TonySUGI31_20060403.pdf). If the real time and the CPU time vary greatly and these times should be similar in your environment, find out what is causing the difference.

If you think that you have a hardware issue, see "A Practical Approach to Solving Performance Problems with the SAS System" at [support.sas.com/rnd/scalability/papers/solve\\_perf.pdf](https://support.sas.com/rnd/scalability/papers/solve_perf.pdf). This paper provides information about what to look for and what investigative tools to use.

If you determine that your hardware is properly configured, then next look at what the SAS code is doing. Transformations generate SAS code. Understanding what this code is doing is very important to ensure that you do not duplicate tasks, especially SORTs, which are very resource-intensive. For details about sorts, see the section "Sorting Data".

The ultimate goal is to configure the hardware so that you can load your data in the desired time frame by avoiding needless I/O in the ETL flows.

## SAS Options for Analyzing Job Performance

To analyze performance, it is recommended that you add the following SAS options to capture detailed information in the log about what the SAS tasks are doing:

```
FULLSTIMER
```

```
MSGLEVEL=I (This option prints additional notes that pertain to indexes, merge processing, sort utilities, and CEDA usage; along with the standard notes, warnings, and error messages.)
```

```
SOURCE, SOURCE2
```

```
MPRINT
```

```
NOTES
```

To interpret the output from the FULLSTIMER option, see "A Practical Approach to Solving Performance Problems with the SAS System" at [support.sas.com/rnd/scalability/papers/solve\\_perf.pdf](http://support.sas.com/rnd/scalability/papers/solve_perf.pdf).

In addition to the preceding options, the following SAS statements will echo useful information to the SAS log:

```
PROC OPTIONS OPTION=UTILLOC; run;
```

```
PROC OPTIONS GROUP=MEMORY; run;
```

```
PROC OPTIONS GROUP=PERFORMANCE; run;
```

```
LIBNAME _ALL_ LIST;
```

There are hundreds of SAS options, and the PROC OPTIONS statement lists all the SAS options and their current settings in the SAS log. If you want to see the value that was specified for the SAS MEMORY option, issue the PROC OPTIONS statement with the GROUP=MEMORY parameter. If you want to see only the SAS options that pertain to performance, issue the PROC OPTIONS statement with the GROUP=PERFORMANCE parameter.

The LIBNAME \_ALL\_ LIST statement lists to the SAS log information (physical path location, engine being used, and so on) regarding each libref that is currently assigned to the SAS session. This information is helpful for understanding which file systems on the computer are being used during the ETL flow.

For more information, see the section "SAS Invocation Options for SAS Data Integration Studio Jobs and Transformations".

## Re-Direct Large SAS Logs to a File

Each SAS Data Integration Studio job generates or retrieves SAS code that reads sources and creates targets in physical storage. The SAS log for a job provides critical information about what happened when the job executed. However, large jobs can create large logs, which can slow performance, considerably. In order to avoid this problem, you can re-direct the SAS log to a permanent file, then turn off logging using the **Log** tab in the Process Designer window. For more information about re-directing logs to a file for each job run in SAS Data Integration Studio, see the section "SAS Logs for Analyzing ETL Flows". Also, see the section "Administering SAS Data Integration Studio, Redirecting Output and Logging Information to a File" in *SAS OnlineDoc 9.1.3* at [support.sas.com/onlinedoc/913/docMainpage.jsp](http://support.sas.com/onlinedoc/913/docMainpage.jsp).

Alternatively, you can add the following code on the **Pre and Post Process** tab in the Properties window for a job:

```
proc printto log=...<path_to_log_file> NEW; run;
```

The preceding code re-directs the log to the specified file. When you specify this log file, be sure to use the appropriate host-specific syntax of the host that your job will run on, and verify that you have Write access to the location that the log will be written to.



**Note:** Consider re-directing the log file to a library that is not heavily used by your system so that the creation of the log file does not impact performance.

For details about adding a pre-process to a SAS Data Integration Studio job, see the section "Add SAS Code on the Pre- and Post-Processing Tab".

## View or Hide the Log in SAS Data Integration Studio

The Process Designer window in SAS Data Integration Studio has a **Log** tab that displays the SAS log for the job that is in the window. To display or hide the **Log** tab, perform the following tasks:

1. From the SAS Data Integration Studio desktop, select **Tools ► Options** to open the Options window.
2. In the Options window, click the **General** tab.
3. Click or unclick the check box that controls whether the **Log** tab is displayed in the Process Designer window.

## Debugging Transformations in an ETL Flow

If you are analyzing a SAS Data Integration Studio job and the information that is added by the logging options is not enough for your purposes, consider adding temporary debugging steps to your ETL flows to generate additional information. Here are some suggestions for doing this:

- Add a Return Code Check transformation.

For details about using the Return Code Check transformation, see the SAS Data Integration Studio online help. For information about adding custom debugging steps, including the addition of user-written transformations, see the section "Add Your Own Code to User-Written Code Transformation".

- Add a User-Written transformation and write your own code.

You can use the User-Written and Return Code Check transformations together or use them separately to direct information to the log or to alternate destinations such as external files, tables, or e-mail. Possible uses of User-Written transformations include: e-mailing status values; testing frequency counts; listing SAS macro variable settings, or listing the run-time values of system options. An advantage of using these two transformations is that they can be inserted between existing transformations and removed later without affecting the mappings in the original ETL flow.

When you are working with the SQL Join transformation version 2, enable the Debug property on the transformation by typing the letter 'Y' or clicking **Yes** on the screen in the Value column (Figure 5). The Debug property enables trace information and other useful debugging information to be sent to the log for this transformation.

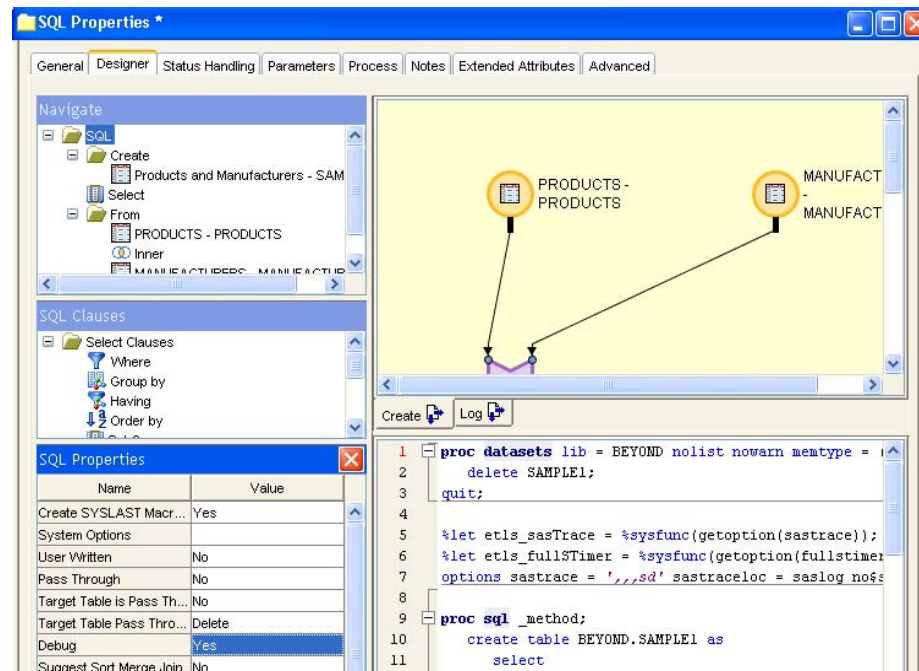


Figure 5. Debug Property Turned On in Version 2 of the SQL Join Transformation

---

## Transformation Output Tables Used to Analyze ETL Flows

Most transformations (especially data transformations) in a SAS Data Integration Studio job create at least one output table and, by default, store that table in the WORK library on the SAS Workspace Server that executes the job. The output table for each transformation becomes the input to the next transformation in the ETL flow. All output tables are deleted when the job is finished or the current server session ends.

If a job is not producing the expected output or if you suspect that something is wrong with a particular transformation, you can view the output tables for the transformations in the job and verify that each transformation is creating the expected output.



**Tip:** In addition to being useful when analyzing ETL flows, output tables can be preserved to determine how much disk space they require or to re-start an ETL flow after it has failed at a particular step (transformation).

### Viewing the Output File for a Transformation

In SAS Data Integration Studio, you can use the View Data window to display the contents of a transformation's output table. To view the output file, perform the following tasks:

1. Open the job in the Process Designer window.
2. Submit the job for execution. The job—or at least the relevant transformations—must execute successfully. (Otherwise, a current output table would not be available for viewing.)
3. Right-click the transformation of the output table that you want to view, and select **View Data** from the pop-up menu. The transformation's output table is displayed in the View Data window.

This approach will work if you do not close the Process Designer window. When you close the Process Designer window, the current server session ends, and the output tables are deleted.

### SAS Options That Preserve Files Created during Batch Jobs

When SAS Data Integration Studio jobs are executed in batch mode, a number of SAS options can be used to preserve intermediate files in the WORK library. These system options can be set as described in the section "SAS Invocation Options for SAS Data Integration Studio Jobs and Transformations".

Use the `NOWORKINIT` system option to tell SAS not to create a new WORK sub-directory at invocation (use the most recent existing WORK sub-directory). Use the `NOWORKTERM` system option to prevent SAS from erasing the current WORK sub-directory at termination. For example, to create a permanent SAS WORK library in UNIX and Windows operating environments, start the SAS Workspace Server with the `WORK` option to re-direct the WORK files to a permanent WORK library. The `NOWORKINIT` and `NOWORKTERM` system options must be included, as shown here.

```
C:\>"C:\Program Files\SAS\SAS 9.1\sas.exe" -work "C:\Documents and
Settings\sasapb\My Documents\My SAS Files\My SAS Work Folder" -
noworkinit

-noworkterm
```

For more information about the `NOWORKINIT` and `NOWORKTERM` system options, see the *SAS OnlineDoc 9.1.3* available at [support.sas.com/onlinedoc/913/docMainpage.jsp](http://support.sas.com/onlinedoc/913/docMainpage.jsp).

In the following example, the generated `WORK` files are re-directed to the folder **My SAS Work Folder**.

To create a permanent SAS `WORK` library in the z/OS operating environment, edit your JCL (Job Control Language) statements by changing the `WORK DD` statement to a permanent MVS data set. For example:

```
//STEP1 EXEC SDSSAS9,REGION=50M  
  
//* changing work lib definition to a permanent data set  
  
//SDSSAS9.WORK DD DSN=userid.somethin.sasdata,DISP=OLD  
  
//* other file defs  
  
//INFILE DD ...
```



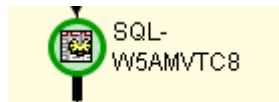
**Caution:** If you re-direct `WORK` files to a permanent library, you must manually delete these files to avoid running out of disk space.

## Re-Directing Output Files for a Transformation

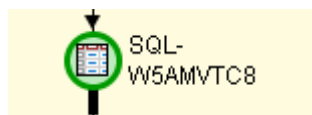
The default name for a transformation's output table is a two-level name that specifies the `WORK` libref and a generated member name, such as `work.w5AMVTC8`. Transformation output tables are visible in the Process Editor and have a Property Sheet that contains the physical name and location of the table. By default, the table is created in the `WORK` library. To change the location for the table, open the Property Sheet for the table, and specify an alternate library for the table on the **Physical Storage** tab. The location of the output table can be a SAS library or a relational **database library**. Re-directing the file has the added benefit of providing you with the ability to specify which output tables you want to retain and allow the rest of the tables to be deleted, by default. You can use this approach as a methodology for checkpoints by writing specific output tables to disk when needed.

Some transformations can create either a table or a view as their output table. This is also specified on the output table if the output table is a `WORK` table. To specify whether a transformation should create a table or a view by default, right-click the transformation's output table, and click or unclick the **Create View** option. A visual indication of the type of output (table or view) that the transformation will generate is indicated on the icon of the `WORK` table in the Process Editor window.

The transformation creates a view.



The transformation creates a table.



If you specify an output table by using a single-level name (for example, EMPLOYEE), instead of a two-level name (for example, WORK.EMPLOYEE), SAS automatically sends the output table into the User library, which defaults to the WORK library. However, this default behavior can be changed by any SAS user via the USER= system option, which enables the user to re-direct the User library to a different library. If the USER= system option is set, single-level tables are stored in the User library that has been re-directed to a different library, instead of being stored in the WORK library.



**Note:** Changing the property setting of the transformation output to create a view does not always guarantee that a view will be created because some SAS procedures, by default, cannot create a view. One example of a transformation that will create a view is the SAS Sort transformation, which will create a view however and wherever possible.

## List Data Transformation Added to the ETL Flow

In SAS Data Integration Studio, you can use the List Data transformation to print the contents of an output table from the preceding transformation in an ETL flow. Add the List Data transformation after any transformation output table that you want to see.

The List Data transformation uses the PRINT procedure to produce output. Any options that are associated with PROC PRINT can be added on the **Options** tab in the transformation's Property window. By default, output goes to the **Output** tab of the Process Designer window. Output can also be directed to an HTML file. For large data, customize this transformation to print just a subset of the data. For details, see the section "Limit Transformation's Input".

## User-Written Code Transformation Added to the ETL Flow

You can add a User-written Code transformation to the end of an ETL flow that would move or copy some of the data sets in the WORK library to a permanent library.

For example, assume that there are three tables in the WORK library (test1, test2, and test3). The following code moves the three tables from the WORK library to a permanent library named PERMLIB, and then deletes the tables in the WORK library.

```
libname permlib base

      "C:\Documents and Settings\ramich\My Documents\My SAS Files\9.1";

proc copy move

  in = work

  out = permlib;

  select test1 test3;

run;
```

For information about user-written transformations, see the section "Add Your Own Code to User-Written Code Transformation".

























































































































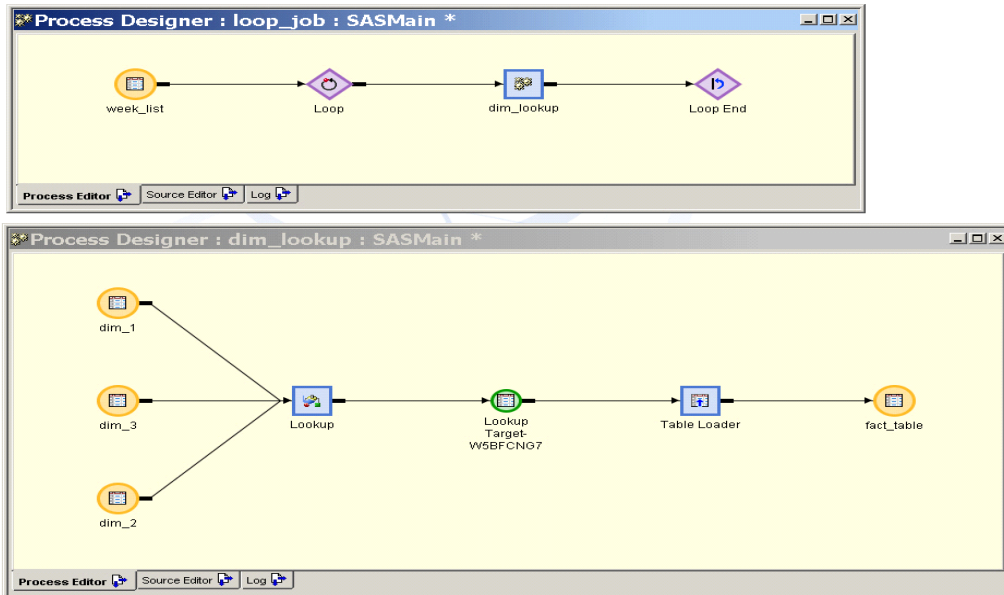


Figure 16. SAS Data Integration Studio Dimension Lookup

#### Throttle Settings in the Loop Transformation

Figure 17 shows the settings that were used in SAS Data Integration Studio on the loop transformation to control (dictate) the method of execution. Notice that "Run all processes concurrently" is selected. "Wait for all processes to complete before continuing" was also selected. This specifies that loop sub-tasks should be complete before continuing to the next step in the master ETL process. By using these settings, the SAS code determines the number of concurrent processes to be launched.

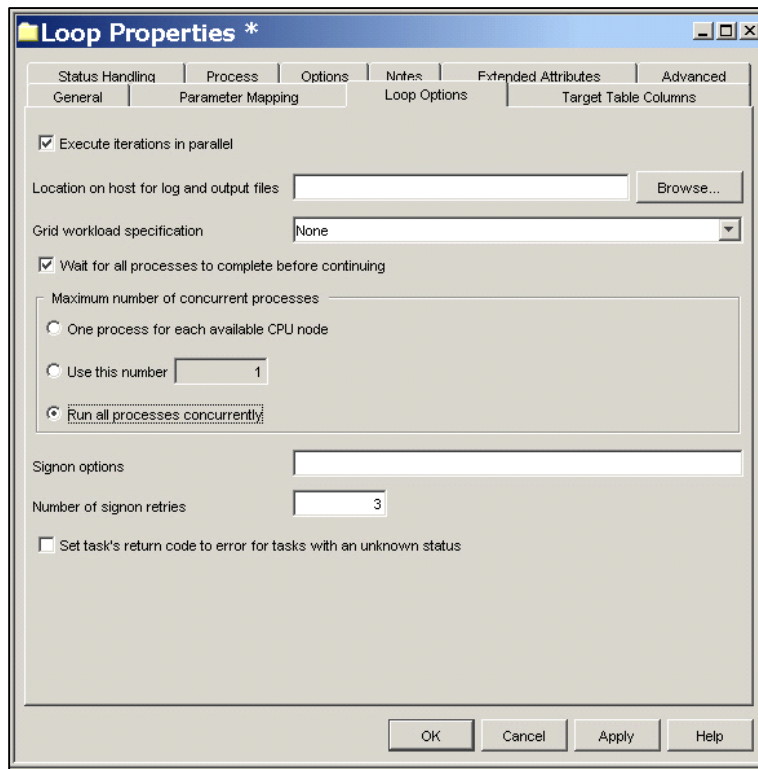


Figure 17. SAS Data Integration Studio Loop Transformation Settings

#### Throttle Settings Used in the Scheduler

In the set-up run, Platform LSF software controls the speed at which these jobs were launched. Platform LSF has the capability to control the speed at which jobs are submitted to the operating system based on various parameters (resources available, max number of jobs to run at one time, how many jobs to run on a particular grid node, and so on).

The following list shows some of the Platform LSF settings that were used on a very large UNIX implementation during high volume ETL runs that involve multiple terabytes of data and many concurrent processes. These settings throttle the number of jobs that are launched in order to stagger the start of tasks to prevent resource contention during job startup.

```

MBD_SLEEP_TIME = 5

SBD_SLEEP_TIME = 30

JOB_SCHEDULING_INTERVAL = 0

JOB_ACCEPT_INTERVAL = 3

NEW_JOB_SCHED_DELAY = 0

```

For details about the settings to use for your task, see *Administering Platform LSF Version 6.0 - January 2004* at [ftp.sas.com/techsup/download/unix/lsf\\_admin\\_6.0.pdf](http://ftp.sas.com/techsup/download/unix/lsf_admin_6.0.pdf).

It is recommended that you, initially, keep the default settings in Platform LSF in the LSF configuration directory until some basic performance testing is completed. Monitor the jobs to see how fast or how slowly they are launching, and then adjust Platform LSF as needed. Platform LSF settings vary greatly based on the hardware, software setup, and job task.

### Scenario Results

Figure 18 shows the results of running the Star Schema scenario across multiple CPU cores. As mentioned earlier, the dimension table builds in phase I were not converted to run in parallel beyond building each dimension table as a separate process. Therefore, in phase I, the maximum number of jobs that ran in parallel was five, which coincides with the number of dimension tables that were built. Phase II of the scenario consisted of the dimension lookup and parallel fact table load for each week of retail transaction data. By using the weekly transaction file as the partition for parallel processes, up to 260 simultaneous processes could be run. However, due to resource restraints and contention, not all 260 processes were executed concurrently. In the Star Schema scenario, Platform LSF software scheduled (throttled) the start of each job in order to best utilize available resources.

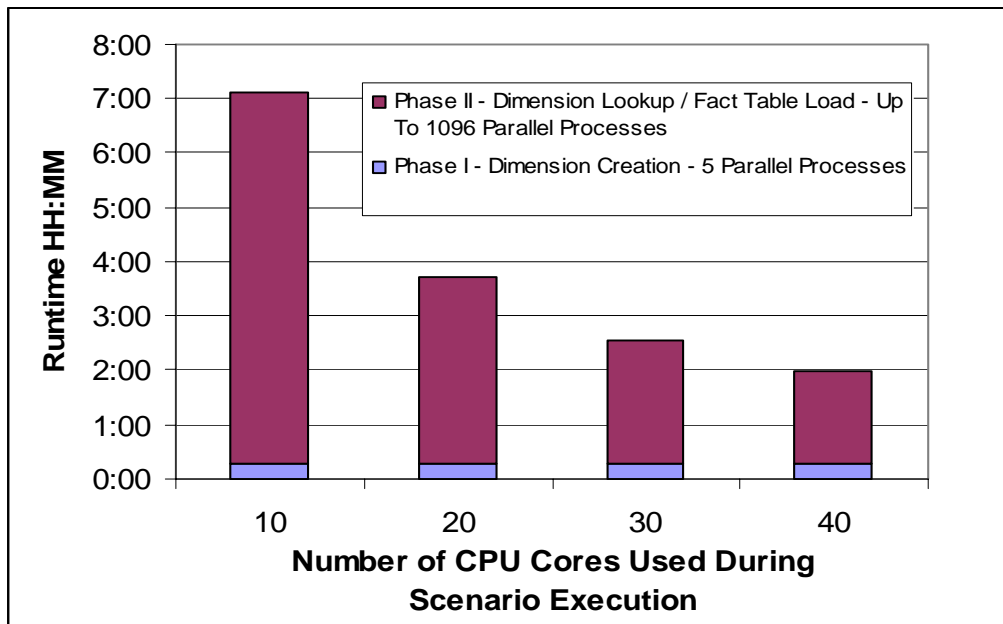


Figure 18. Results from Running the Star Schema Parallel Build across Multiple CPU Cores

Looking at the results shown in Figure 18, it is easy to see that changing from 10 CPUs to 20 CPUs resulted in a major improvement in performance (50% run-time reduction). Run time was further reduced as more CPUs (over 20) were added, but the return on investment was less significant (this was attributed to data partitioning overhead, job design, and system architecture). Results can vary greatly based on the ETL job design and the resources that are available.

It is important to note that any decrease in run time can be advantageous. Therefore, adding more system resources (that is, CPUs, memory, disk) despite decreasing returns on investment, can be very valuable to an enterprise. Investment cost tolerance varies greatly depending on the business problem.

---

## Recommended Reading

Bartucca, Frank and Edward Hayes-Hall. 2005. "Achieving Optimal I/O Performance with SAS®9", *Proceedings of the 30<sup>th</sup> Annual SUGI Conference*. Cary, NC: SAS Institute Inc. Available at [www2.sas.com/proceedings/sugi30/221-30.pdf](http://www2.sas.com/proceedings/sugi30/221-30.pdf).

Karp, Andrew H. and David Shamlin. 2003. "Indexing and Compressing SAS® Data Sets: How, Why, and Why Not". *Proceedings of the 28<sup>th</sup> Annual SUGI Conference*. Cary, NC: SAS Institute Inc. Available at [www2.sas.com/proceedings/sugi28/003-28.pdf](http://www2.sas.com/proceedings/sugi28/003-28.pdf).

Karp, Andrew. 2000. "Indexing and Compressing SAS® Data Sets: How, Why, and Why Not". *Proceedings of the 25<sup>th</sup> Annual SUGI Conference*. Cary, NC: SAS Institute Inc. Available at [www2.sas.com/proceedings/sugi25/25/aa/25p005.pdf](http://www2.sas.com/proceedings/sugi25/25/aa/25p005.pdf).

Platform Computing Corporation. 2004. *Administering Platform LSF Version 6.0*. Available at [ftp.sas.com/techsup/download/unix/lsf\\_admin\\_6.0.pdf](http://ftp.sas.com/techsup/download/unix/lsf_admin_6.0.pdf).

SAS Institute Inc. 2006. *SAS/ACCESS®9.1.3 for Relational Databases: Reference*. Available at [support.sas.com/documentation/onlinedoc/91pdf/sasdoc\\_913/access\\_rdbref\\_9297.pdf](http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/access_rdbref_9297.pdf).

SAS Institute Inc. 2006. *SAS® 9.1.3 Intelligence Platform Planning and Administration Guide*. Available at [support.sas.com/documentation/configuration/iaplanning913.pdf](http://support.sas.com/documentation/configuration/iaplanning913.pdf).

SAS Institute Inc. 2006. *SAS® Data Integration Studio 3.3: User's Guide*. Available at [support.sas.com/documentation/onlinedoc/etls/usage33.pdf](http://support.sas.com/documentation/onlinedoc/etls/usage33.pdf).

SAS Institute Inc. 2006. *SAS® Intelligence Platform: Overview*. Available at [support.sas.com/documentation/configuration/biov.pdf](http://support.sas.com/documentation/configuration/biov.pdf).

SAS Institute Inc. 2006. *SAS OnlineDoc® 9.1.3*. Available at [support.sas.com/onlinedoc/913/docMainpage.jsp](http://support.sas.com/onlinedoc/913/docMainpage.jsp).

SAS Institute Inc. 2006. *SAS® Scalable Performance Data Server® 4.4: User's Guide*. Available at [support.sas.com/documentation/onlinedoc/91pdf/sasdoc\\_913/scalable\\_ug\\_9722.pdf](http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/scalable_ug_9722.pdf).

Seckosky, Jason. 2004. "The DATA Step in SAS 9: What's New?". *Proceedings of the 29<sup>th</sup> Annual SUGI Conference*. Cary, NC: SAS Institute Inc. Available at [support.sas.com/rnd/base/topics/datastep/dsv9-sugi-v3.pdf](http://support.sas.com/rnd/base/topics/datastep/dsv9-sugi-v3.pdf).

Shoemaker, Jack. 2001. "Eight PROC FORMAT Gems". *Proceedings of the 26<sup>th</sup> Annual SUGI Conference*. Cary, NC: SAS Institute Inc. Available at [www.suk.sas.com/proceedings/26/26/p062-26.pdf](http://www.suk.sas.com/proceedings/26/26/p062-26.pdf).

---

# Glossary

**business key**

one or more columns in a dimension table that comprise the primary key in a source table in an operational system. See also: **dimension table**, **primary key**.

**data cleansing**

the process of eliminating inaccuracies, irregularities, and discrepancies from character data.

**database library**

a collection of one or more database management system files that are recognized by SAS and that are referenced and stored as a unit. Each file is a member of the library.

**debug**

a process of eliminating errors in code by examining logs and data to determine where or if an error has occurred.

**derived mapping**

a mapping between a source column and a target column in which the value of the target column is a function of the source column. For example, if two tables contain a Price column, the value of the target table Price column might equal the value of the source table Price column times 0.8.

**dimension**

a category of contextual data or detail data that is implemented in a data model such as a star schema. For example, in a star schema, a dimension named Customers might associate customer data with transaction identifiers and transaction amounts in a fact table. See also: **fact table**.

**dimension table**

in a star schema or a snowflake schema, a table that contains data about a specific dimension. A primary key connects a dimension table to a related fact table. For example, if a dimension table that is named Customers has a primary key column that is named Customer ID, then a fact table that is named Customer Sales might specify the Customer ID column as a foreign key. See also: **fact table**, **foreign key**, **primary key**.

**ETL flow**

a collection of SAS tasks that perform an extract, transform, and load of data.

**fact table**

the central table in a star schema or a snowflake schema. Usually, a fact table contains numerical measurements or amounts and is supplemented by contextual information in dimension tables. For example, a fact table might include transaction identifiers and transaction amounts. Dimension tables add contextual information about customers, products, and sales personnel. Fact tables are associated with dimension tables via key columns. Foreign key columns in the fact table contain the same values as the primary key columns in the dimension tables. See also: **dimension table**, **foreign key**.

**foreign key**

one or more columns that are associated with a primary key or a unique key in another table. A table can have one or more foreign keys. A foreign key is dependent upon its associated primary or unique key; a foreign key cannot exist without that primary or unique key. See also: **primary key**, **unique key**.

**generated key**

a column in a dimension table that contains values that are sequentially generated using a specified expression. Generated keys implement surrogate keys and retained keys. See also: **dimension table**, **surrogate key**, **retained key**.

**generated transformation**

in SAS Data Integration Studio, a transformation that is created with the Transformation Generator wizard, which helps you specify SAS code for the transformation. See also: **transformation**.

**job**

a collection of SAS tasks that specifies processes that create output.

**metadata object**

specifies an instance of a metadata type such as the metadata for a particular data store or metadata repository. All SAS Open Metadata Interface clients use the same methods to read or write a metadata object, whether the object defines an application element or a metadata repository. See also: **metadata repository**.

**metadata repository**

a collection of related metadata objects such as the metadata for a set of tables and columns that are maintained by an application. A SAS Metadata Repository is an example. See also: **metadata object**.

**operational system**

one or more programs (often, relational databases) that provide source data for a data warehouse.

**primary key**

one or more columns that uniquely identify a row in a table. A table can have only one primary key. Columns in a primary key cannot contain null values.

**process flow diagram**

a diagram in the Process Editor that specifies the sequence of each source, target, and process in a job. Each process in the diagram is called a transformation step. See also: **transformation**.

**register**

saves metadata about an object to a metadata repository. For example, if you register a table, you save metadata about that table to a metadata repository. See also: **metadata repository**.

**SAS Application Server**

a server that provides SAS services to a client. In SAS Open Metadata Architecture, the metadata for a SAS Application Server can specify one or more server components that provide SAS services to a client. See also: **SAS Open Metadata Architecture**.

**SAS Management Console**

a Java application that provides a single user interface for performing SAS administrative tasks.

**SAS Open Metadata Architecture**

a general-purpose metadata management facility that provides metadata services to SAS applications. SAS Open Metadata Architecture enables applications to exchange metadata, which makes it easier for these applications to work together.

**server component**

in SAS Management Console, metadata that specifies how to connect to a particular type of SAS server on a particular computer. See also: **SAS Management Console**.

**source**

an input to an operation.

**star schema**

tables in a database in which a single fact table is connected to multiple dimension tables. This is visually represented in a star pattern. SAS OLAP cubes can be created from a star schema. See also: **dimension table, fact table**.

**surrogate key**

a numeric column in a dimension table that is the primary key of that table. The surrogate key column contains unique integer values that are generated sequentially when rows are added and updated. In the associated fact table, the surrogate key is included as a foreign key in order to connect to specific dimensions. See also: **dimension table, fact table, foreign key, primary key**.

**target**

an output of an operation.

**thrashing**

resource contention under heavy utilization

**transformation**

a process that specifies how to extract data, transform data, or load data into data stores. Each transformation that you specify in a process flow diagram generates or retrieves SAS code. You can specify user-written code in the metadata for any transformation in a process flow diagram. See also: **process flow diagram**.

**unique key**

one or more columns that uniquely identify a row in a table. A table can have one or more unique keys. Unlike a primary key, a unique key can contain null values. See also: **primary key**

Place holder for copyright statement