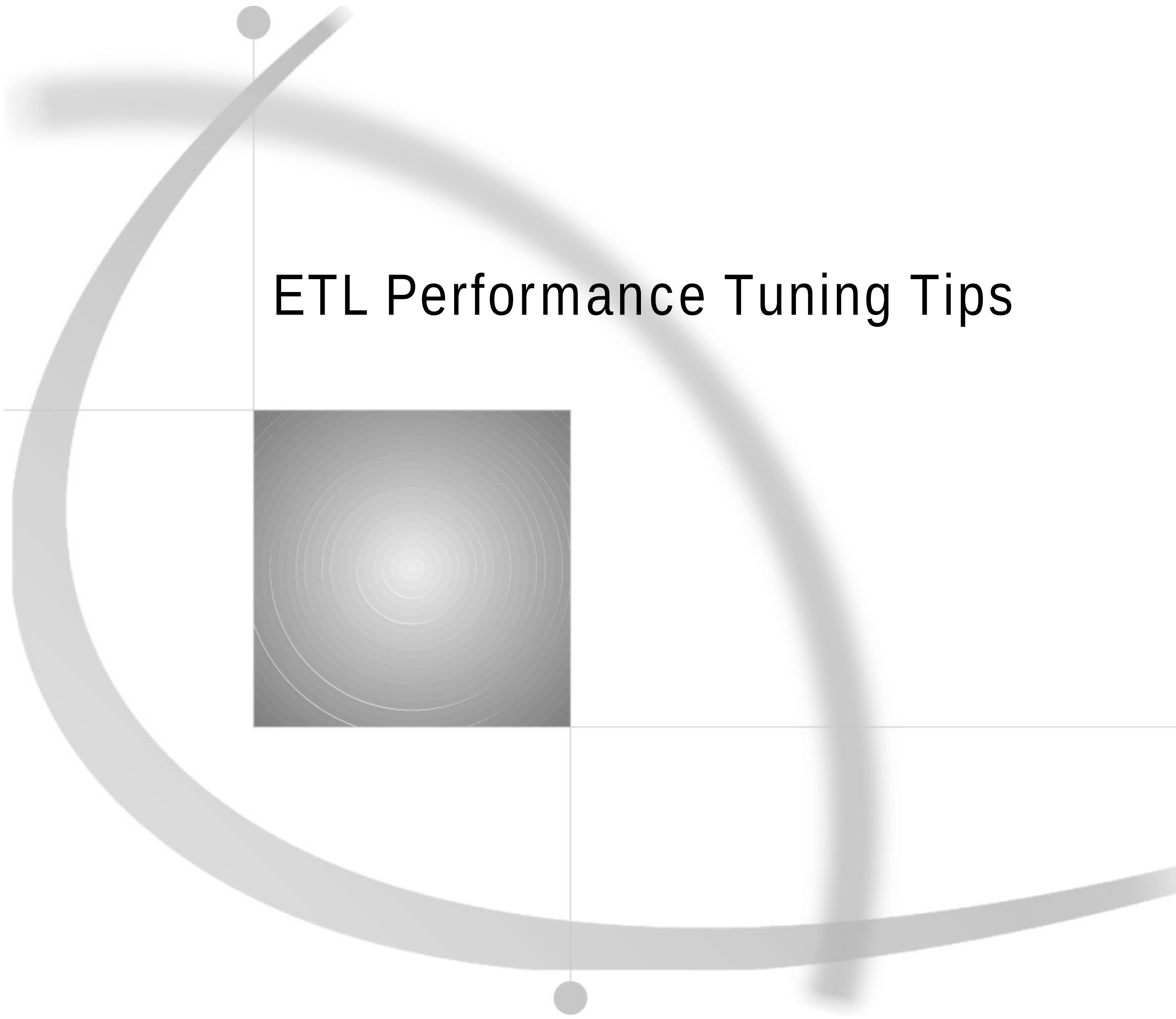
ETL Performance Tuning Tips



THE POWER TO KNOW.

ETL Performance Tuning Tips

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.

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.

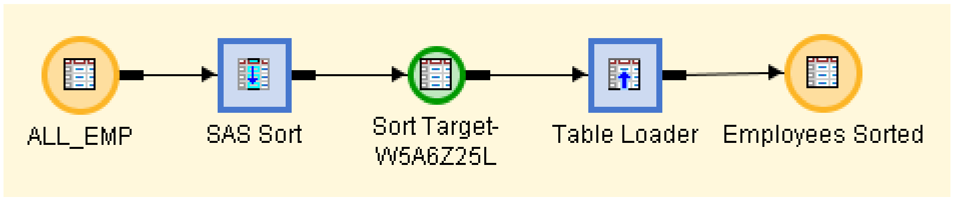


Figure 1. SAS Data Integration Studio Process Flow Diagram for a Job That Sorts Data

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 Trabsformations".

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.

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) 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.

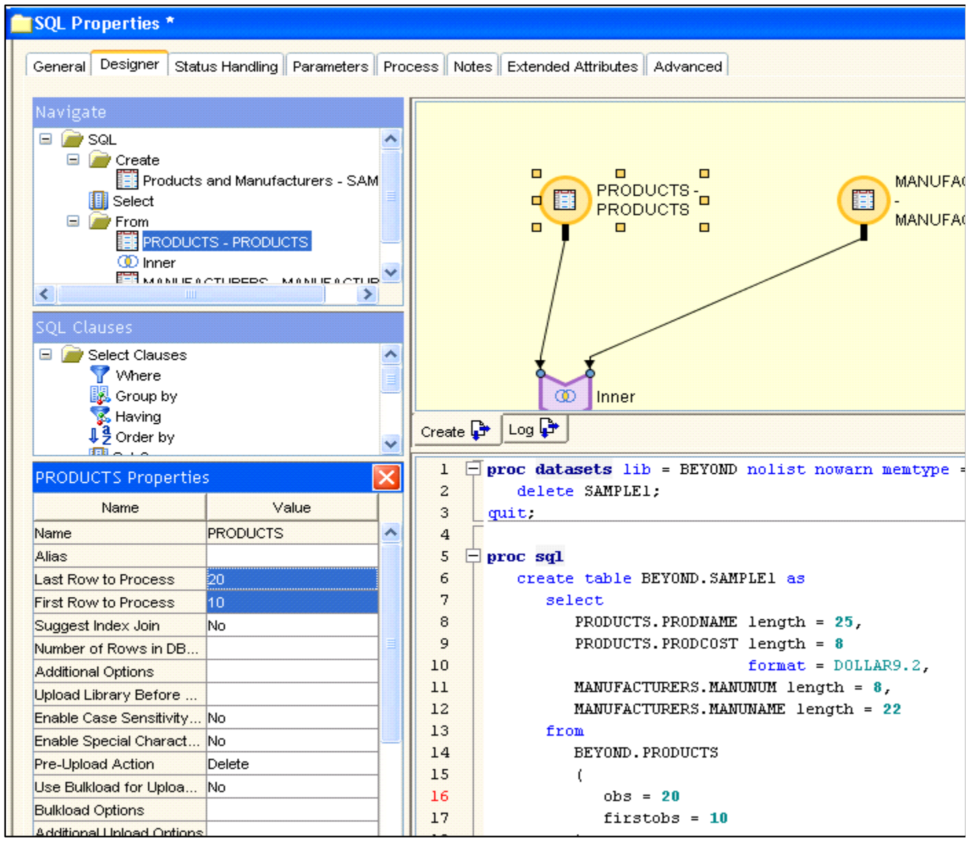
  In SAS Data Integration Studio, you can do this with a simple edit in the source editor.

  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.

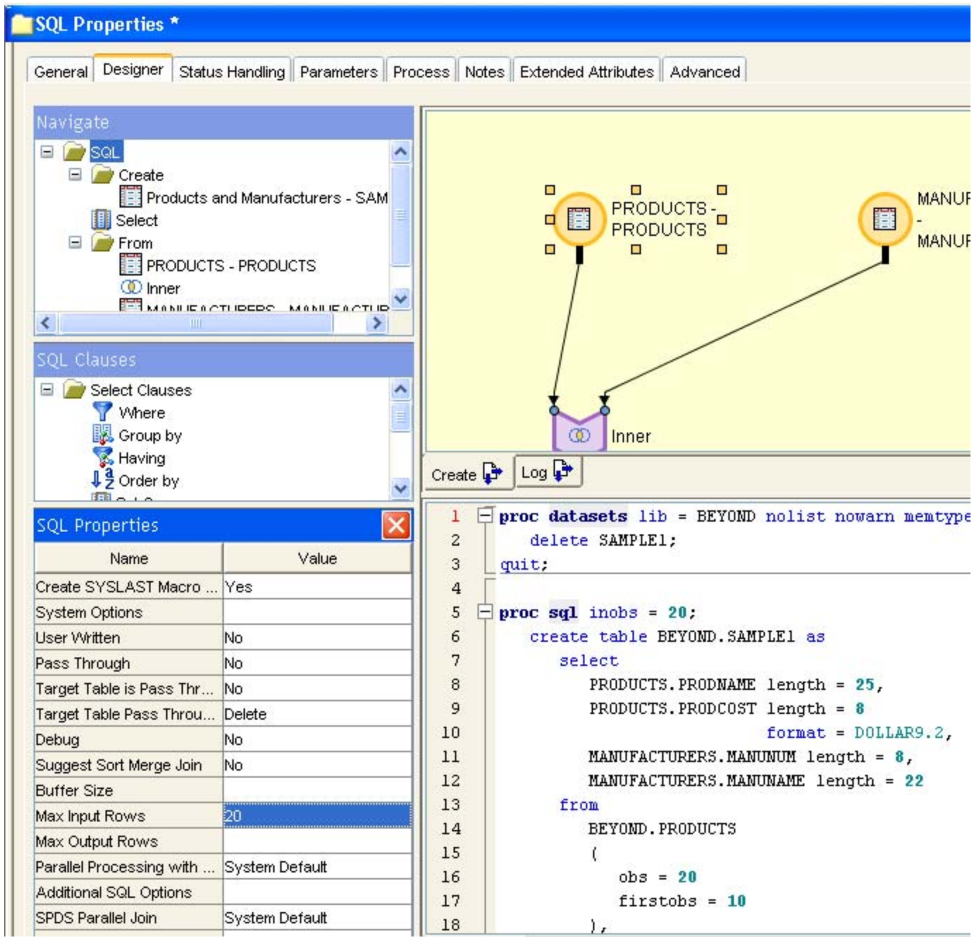


Name	Value
Name	PRODUCTS
Alias	
Last Row to Process	20
First Row to Process	10
Suggest Index Join	No
Number of Rows in DB...	
Additional Options	
Upload Library Before ...	
Enable Case Sensitivity...	No
Enable Special Charact...	No
Pre-Upload Action	Delete
Use Bulkload for Uploa...	No
Bulkload Options	

```
proc datasets lib = BEYOND nolist nowarn memtype =
    delete SAMPLE1;
quit;

proc sql
    create table BEYOND.SAMPLE1 as
       select
          PRODUCTS.PRODNAME length = 25,
          PRODUCTS.PRODCOST length = 8
                           format = DOLLAR9.2,
          MANUFACTURERS.MANUNUM length = 8,
          MANUFACTURERS.MANUNAME length = 22
       from
          BEYOND.PRODUCTS
          (
             obs = 20
             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.



SQL Properties

Name	Value
Create SYSLAST Macro ...	Yes
System Options	
User Written	No
Pass Through	No
Target Table is Pass Thr...	No
Target Table Pass Throu...	Delete
Debug	No
Suggest Sort Merge Join	No
Buffer Size	
Max Input Rows	20
Max Output Rows	
Parallel Processing with ...	System Default
Additional SQL Options	
SPDS Parallel Join	System Default

```
proc datasets lib = BEYOND nolist nowarn memtype
   delete SAMPLE1;
quit;

proc sql inobs = 20;
   create table BEYOND.SAMPLE1 as
      select
         PRODUCTS.PRODNAME length = 25,
         PRODUCTS.PRODCOST length = 8
                     format = DOLLAR9.2,
         MANUFACTURERS.MANUNUM length = 8,
         MANUFACTURERS.MANUNAME length = 22
      from
         BEYOND.PRODUCTS
         (
            obs = 20
            firstobs = 10
         ),
```

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:

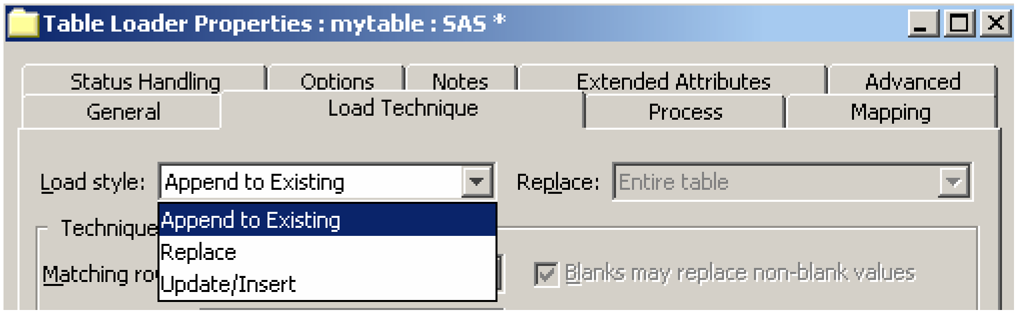
```
options obs=MAX;
```

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.



Load style: Append to Existing

- Append to Existing
- Replace
- Update/Insert

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. 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. 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.

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.

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:

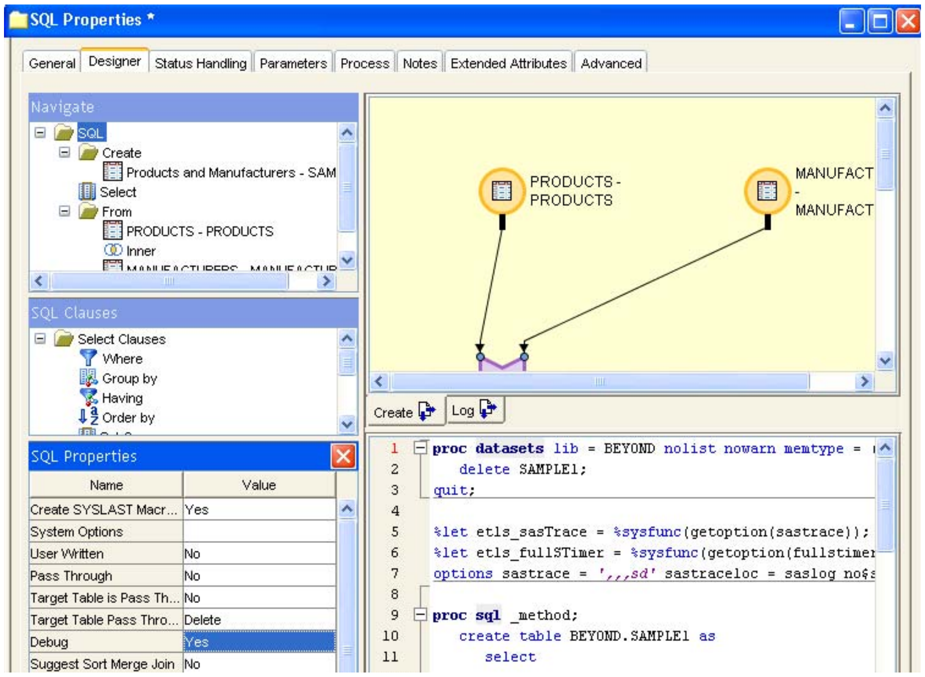
- Add a Return Code Check transformation.

  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.



Name	Value
Create SYSLAST Macr...	Yes
System Options	
User Written	No
Pass Through	No
Target Table is Pass Th...	No
Target Table Pass Thro...	Delete
Debug	Yes
Suggest Sort Merge Join	No

```
proc datasets lib = BEYOND nolist nowarn memtype =
   delete SAMPLE1;
quit;

%let etls_sasTrace = %sysfunc(getoption(sastrace));
%let etls_fullSTimer = %sysfunc(getoption(fullstimer
options sastrace = ',,,sd' sastraceloc = saslog no$s

proc sql _method;
   create table BEYOND.SAMPLE1 as
      select
```

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.

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 ….
```

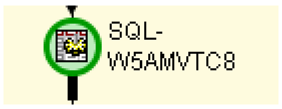
Caution: If you re-direct WORK files to a permanent library, you must manually delete these files to avoid running out of disk space.

Re-Directing Output Files for a Transformation

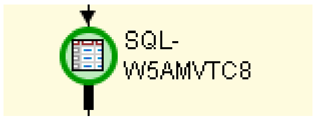
The default name for a transformation's output table is a two-level name that specifies the WORK libref and a generated member name, such as `work.W5AMVTC8`. Transformation output tables are visible in the Process Editor and have a Property Sheet that contains the physical name and location of the table. 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 theProcess 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 eanbles 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".

Simple Tuning of ETL Flows Results in Immediate Gains

When you are writing and deploying ETL flows, there are some areas that, when tuned, can give you immediate performance gains. Here are some simple ways that you should consider when you write ETL flows or run into problems when you deploy ETL flows.

Sorting Data

Sorting is a common and resource-intensive component of ETL. Sorts occur explicitly as PROC SORT steps and implicitly in other operations such as joins. Effective sorting requires a detailed analysis of performance and resource usage.

Sorting is either handled by SAS or is passed to an external sort engine. There are several types of external sort engines: third-party sort utilities, relational databases, and the SAS Scalable Performance Data Server (SAS SPD Server). SAS does not, automatically, pass sorting to third-party sort utilities such as SyncSort and DFSORT. Under most circumstances, SAS passes sorting to the relational database or the SAS SPD Server. When you use PROC SORT, you can alter this behavior and specify SAS-based sorting by using the `SORTPGM=` option. In other cases (for example, when you use ORDER BY clauses), SAS always passes sorting to the relational database or the SAS SPD Server. If you are using PROC SORT, the option `MSGLEVEL=I` verifies whether the sort is performed by SAS or by an external engine. The following note in the log "`Sorting was performed by the data source`" indicates that an external engine was used.

The information in this section describes SAS-based sorting except where noted.

Note: It is not a good practice to sort data and output the data to a relational database because the sort order is lost. Therefore, the sort was a useless expenditure of resources.

Sorting physically re-arranges the rows in a table and orders the rows on one or more key columns. To sort a table, SAS reads the first 'n' table rows into memory until the value of the `SORTSIZE=` option is reached. The 'n' rows are sorted in memory, then output to disk as a "run" in the SORT procedure utility file. Groups of 'n' rows are processed this way until all rows are read, sorted in memory, and output as "runs". Then, the runs are merged to produce the fully sorted table.

When the value for `SORTSIZE=` is large enough to fit all rows in memory at one time, the sorting completes without the need for "runs" or a SORT procedure utility file. This is called an **internal sort**. The performance of internal sorts is optimal if the memory that is required does not exceed the available physical memory.

Note: If the value for `SORTSIZE=` is larger than the available physical memory, system paging can occur and the performance of the sort might degrade when memory resources are exhausted. This issue can become critical on multi-user systems. A best practice in a large ETL environment is to restrict `SORTSIZE=`, which forces most sorts to be external as runs in the SORT procedure utility file.

Sorting occurs implicitly with index creation, ORDER BY clauses, SAS SQL joins, and procedure execution that requires ordering. For any caller, the underlying sort engine is the same. Sort callers include, but are not limited to, the DATA step, PROC SORT, PROC SQL, and PROC SUMMARY. Callers begin the sort process by setting any special sort attributes. For example, PROC SORT sets the EQUALS attribute, which preserves the original order of rows that have identical sort key values. Callers next feed rows to sort, and wait for the sorted results. The format of rows that are passed for sorting might vary by caller. For example, SAS SQL might add pad bytes to rows, which increases the overall amount of data that is sorted.

Sorting large SAS tables requires large SORT procedure utility files. When ETL is running on multiple SAS jobs simultaneously, multiple SORT procedure utility files can be active. For these reasons and the reasons mentioned above, tuning sort performance and understanding sort disk-space consumption are critical.

Tuning Sort Performance

Tuning sort performance is both a science and an art. The science of tuning applies tactics that ensure better performance. The art of tuning is experimenting with tactics that might increase performance, but it is also possible that the same tactics will decrease performance.

Performance-enhancing tactics are listed below in order of importance. The tactics that might result in the most performance gains are discussed first.

Note: Tactics in the category of "art" are indicated by the words "Consider This Approach" in bold.

Leveraging the Improved SAS®9 Sort Algorithm

SAS®9 includes an improved SORT algorithm that incorporates threading and data latency reduction algorithms. SAS®9 multi-threaded sort outperforms a SAS 8 sort in almost all cases. However, in cases in which the data to be sorted is almost in sorted order, the `TAGSORT` option in PROC SORT in SAS 8 can outperform the SAS®9 multi-threaded sort. This situation is addressed later in this paper. In general, use the default SAS®9 multi-threaded sort for optimal sort performance.

Minimizing Width of Rows Being Sorted

The smaller the width of each row that is being sorted, the faster a sort completes. For example, compare sorting rows that are 40 bytes wide with sorting rows that are 80 bytes wide. More 40-byte rows fit into memory and are sorted in memory at the same time. Fewer runs are written to the SORT procedure utility file, and the SORT procedure utility file is half as large. CPU and I/O savings are substantial.

Drop unnecessary columns. This is an obvious tactic to minimize row width. If you don’t need columns in the sort, eliminate them before sorting. For details about dropping columns in SAS Data Integration Studio, see the section "Transformation and Column Optimizations".

Minimize pad bytes. In some situations, SAS adds pad bytes between columns that are fed to the sort. For example, pad bytes are added when a SAS data set is referenced through a SAS SQL view. For details, see the section "Disk Space Consumption When Sorting".

Directing Sort Utility Files to Fast Storage Devices

The `-WORK` and `-UTILLOC` invocation options direct SORT procedure utility files to fast, less-used storage devices. If possible, use the `-UTILLOC` invocation option to isolate SORT procedure utility files from other intermediate files. Some procedure utility files are accessed heavily and segregating them from other active files might improve performance. The SORT procedure utility file is a primary example. By default, it resides with other intermediate files in the SAS WORK directory. With the `-UTILLOC` option, it can be directed to a separate disk location. This enables higher I/O rates, thereby reducing sort time.

SAS WORK is set with the `-WORK` option either in a configuration file or at SAS invocation (in the command line). Except in the z/OS operating environment, when the `-UTILLOC` invocation option is **not** set, SORT procedure utility files and other intermediate files co-reside in the SAS WORK location. (For details about the `-UTILLOC` invocation option for the mainframe, see the *SAS 9.1.3 Companion for z/OS* available at support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/base_zoscom_8406.pdf.)

The `-UTILLOC` invocation option provides granular control of the SORT procedure utility file location. Two utility file locations must be listed; however, PROC SORT only uses the second `UTILLOC` location that is specified. For intensive processing, be sure that UTILLOC and SAS WORK are on different file systems.

If your ETL is join-intensive and ample file systems are available, isolate SORT procedure utility files from other intermediate files by specifying the path option. I/O throughput improves for large joins, which creates large SQL temporary files and large SORT procedure utility files. The following SAS command line in a UNIX environment directs the SORT procedure utility file to volume2 and SQL and other intermediate files to volume1.

```
-work /volume1/saswork -utilloc '(/volume9 /volume2/saswork)'
```

Use PROC OPTIONS to verify your `-UTILLOC` invocation option setting:

```
proc options option=utilloc; run;
```

For details about the SORT procedure and the `-UTILLOC` invocation option, see "Getting the Best Performance from V9 Threaded PROC SORT"at support.sas.com/rnd/papers/sugi30/V9SORT.ppt.

Distributing Sort Utility Files for Parallel Jobs across Multiple, Fast File Systems

When you run multiple SAS Data Integration Studio jobs in parallel, you should distribute SORT procedure utility files across multiple fast, less-used file sytems. Because the SORT procedure utility file is critical to overall performance, leverage multiple devices for multiple SAS jobs that perform sorting. Direct the SORT procedure utility file of each job to a different device.

For example, assume that you have two SAS jobs with large sorts. Direct the respective SORT procedure utility files to different devices by using the `–WORK` invocation option:

```
Job 1:

-work /volume1/saswork

Job 2:

-work /volume3/saswork
```

In addition, assume that you have jobs with large SAS SQL joins. You can further optimize I/O by isolating the jobs and the SORT procedure utility files and intermediate files in the jobs by using the `-UTILLOC` invocation option. For example:

```
Job 1:

-work /volume1/saswork -utilloc '(/volume9 /volume2/saswork)'



Job 2:

-work /volume3/saswork -utilloc '(/volume9 /volume4/saswork)'
```

Consider This Approach: If you isolate SORT procedure utility files by using the `–UTILLOC` invocation option, you might want to obtain further optimization by disabling the operating environment read-ahead and file-caching for the utility files. PROC SORT in SAS implements its own internal read-ahead and file-caching algorithms. On a system that has heavy I/O activity, operating environment read-ahead and file caching provide little added value to the SAS algorithms, but often consume I/O bandwidth and memory elsewhere.

For example, on the AIX operating system, if you are using JFS2 files systems, try to increase optimization by enabling AIX direct I/O by using the following:

```
mount –o dio /volume2

mount –o dio /volume4
```

Note: Do not disable read-ahead and file-caching unless you use both the `-WORK` and `-UTILLOC` invocation options to isolate the SORT procedure utility file from other intermediate files. Then, disable read-ahead and file-caching only on UTILLOC, not on WORK. For details about read-ahead and file-caching, see the section "Direct I/O". **Work with your system administrator before you consider using this approach.**

Explicitly Pre-Sorting on Most Common Sort Key

Consider This Approach: ETL processes might arrange a table in sort order, one or multiple times. For large tables in which sort order is required multiple times, look for a common sort order. Consider pre-sorting on the most frequent sort key. For example, if table ACCOUNT is sorted four times, with three sorts on ACCOUNT_KEY, consider pre-sorting once on ACCOUNT_KEY. Look for instances of the table name that appear in PROC SORT steps, SAS SQL joins, ORDER BY clauses, and other operations that might require sorting. Also, use the `MSGLEVEL=I` option to display information in the SAS log to determine where sorts occur.

Changing Default Value for SORTSIZE= Option

When a sort cannot complete in memory, then setting sort memory to a high value (for example, over 512M) might degrade performance even when there is ample physical memory available. This is due to system overhead and hardware cache misses that are associated with virtual-to-physical memory address translation. For large tables, set `SORTSIZE=` to 256M or 512M. Tune these recommended values further based on empirical testing or based on in-depth knowledge of your hardware and operating environment.

For extremely large tables (a billion or more wide rows), set `SORTSIZE=` to 1G or higher. This helps to prevent an undesirable multi-pass merge condition, which is described later in this paper. Preventing multi-pass merges outweighs the additional system overhead that is associated with a value for SORTSIZE= that is larger than 512M (assuming that increased SORTSIZE does not cause a lot of system paging).

Changing Default Value for MEMSIZE= Option

Set the value for the `MEMSIZE=` option to be at least 50% larger than the value for `SORTSIZE=`. In general, do not set the values for `SORTSIZE=` or `MEMSIZE=` greater than the amount of random access memory (RAM) that is available to a SAS session. The amount of RAM a SAS session can access is always smaller than the amount of physical memory that exists because the operating system, file cache, other SAS sessions, and other applications all contend for memory. For example, with 8GB of physical memory and 2GB reserved for the operating system and file cache, a maximum of 6GB is free for SAS sessions and other applications.

An exception, which is not sort-related, is using a SAS procedure that requires more memory than the amount of available RAM. In this case, you must incur system paging for the procedure to complete.

NOSORTEQUALS System Option

Maintaining the relative order of rows is rarely, if ever, an ETL requirement. If maintaining the relative order of rows with identical key values is not important, set the system option `NOSORTEQUALS` to save resources. Alternatively, add the `NOEQUALS` option to applicable PROC SORT steps in your ETL process flow.

UBUFNO Option Maximum of 20

The `UBUFNO` option specifies the number of utility I/O buffers. In some cases, maximizing the value of `UBUFNO` increases sort performance up to 10%. Increasing the value of `UBUFNO` has no negative ramifications.

TAGSORT Option for Nearly Sorted Data

Consider This Approach: The SORT procedure has a `TAGSORT` option. `TAGSORT` is an alternative, SAS 8 sort algorithm that is useful for data that is almost in sort order. However, do not apply the `TAGSORT` option, liberally. The cost of applying `TAGSORT` increases rapidly if data is not pre-sorted. Using the `TAGSORT` option on a large unsorted data set results in extremely high sort times when compared to a SAS®9 multi-threaded sort.

Using the TAGSORT option on nearly-sorted data is most effective when the sort-key width is no more than 5% of the total, uncompressed column width. TAGSORT only pulls key values from the input table, and retains a rowID that points to the original row in the input table. The key-rowID elements sort fast, however, post-sort I/O to extract rows in order from the input table is expensive. On unsorted data, this post-sort extract results in random I/O on the input data and poor overall performance.

For relational database data, SAS options are available that encourage SAS based sorting in place of default relational database-based sorting. If you encourage SAS sorting of relational database data, do not use the TAGSORT option, which will degrade performance in this instance.

Leveraging External Sort Engines for Pre-Sorting Tables if There Are No Data Order Issues

Consider This Approach: If your source data is from a relational database, there might be an opportunity to pre-sort more effectively. For example, pre-sorting in relational databases might outperform SAS sorting. Use options for the SAS Data Integration Studio Extract transformation to generate an ORDER BY clause in the SAS SQL. The ORDER BY clause causes the relational database to return the rows in the specified "sorted" order.

The ORDER BY clause prohibits SAS threaded reads in a relational database. If threaded reads are critical to throughput rates, then pre-sorting in the relational database might be undesirable. SAS automatically threads relational database reads where possible for threaded SAS procedures such as SORT, SUMMARY, REG, DMREG, GLM, and DMINE. To determine if threaded reads occur, use the SASTRACE option that is described in the section "DBSLICEPARM= Data Set Option" in *SAS OnlineDoc 9.1.3* available at support.sas.com/onlinedoc/913/docMainpage.jsp.

For pre-sorting in the relational database, the sort order for SAS and the relational database must match. Missing values (database NULLs) sort differently in SAS than they do in some databases. In those databases, you might restrict relational database pre-sorting to NOT NULL key columns. For example, DB2 and Oracle sort NULL values high, which is different from a sort performed in SAS. However, SQL Server, Teradata, Sybase, and Informix sort NULLs in the same way as SAS. In SAS®9, SAS/ACCESS to Oracle provides the option DEBUG=ORA_SORTNULLS_LIKE_SAS that requests Oracle to sort NULLs in the same way as SAS.

Disk Space Consumption When Sorting

Sorting large SAS tables requires that large procedure utility files are written to the SAS WORK or UTILLOC location. Running SAS sorts in parallel can open multiple sort utility files. Either of these tasks can cause the following problems:

- slower-than-expected sorts due to extra I/O activity
- inability to predict space needs and to size solutions
- out-of-space conditions.

Disk Space Requirements for a Sort

The total space required to complete a sort includes the size of the input data, the size of the sorted output data, and the space needed to sort WORK files. To estimate the total disk space you will need for a sort, multiply the size of the input data by 3. This section provides details on disk space consumption.

SAS includes several aids for sizing, including the DETAILS option in PROC SORT and the column information output by PROC CONTENTS. You will see these aids used in later sections to size the input data, SORT procedure utility file, and output data.

You can also collect sizing numbers with external instrumentation tools. For example, see the section "Disk Space Maximum for Intermediate Files" for a sample shell script to measure the maximum size that is attained by the SORT procedure utility file.

Sizing Input Data

Because sorting is I/O intensive, it's important to begin sorting with only the rows and columns that are needed. The size of the WORK files and the output file for PROC SORT is dependent on the input file size.

When working with compressed data files, PROC SORT uncompresses the rows from the source table and keeps the data in an uncompressed format while the data is manipulated into the specified sort order. This means that the SORT procedure utility files are written to disk in an uncompressed format.

Sizing SORT Procedure Utility Files

When you are trying to size your SORT procedure utility files, you need to consider these factors:

- (for SAS data sets) pad bytes that align each row by 8-bytes

  If you directly sort a SAS data set (no view involved), each row is aligned to an 8-byte boundary unless it is composed only of character columns. For example, a row that has one numeric variable and one $65 character variable (8 + 65 = 73 bytes) consumes 80 bytes. A row that has one numeric variable and one $72 character variable (8 + 72 = 80 bytes) has no alignment overhead.

- 8 bytes per row overhead using EQUALS option

  By default, PROC SORT enables EQUALS processing, which maintains the relative order of rows that have equal key values. The cost results from having to get a rowID that has an additional 8 bytes per row. Specifying the NOEQUALS option in PROC SORT eliminates this overhead.

- per-page, unused space in the SORT procedure utility files

  In SAS®9, sort utility file pages default to 64K per page. Rows do not span pages. Therefore, some space at the end of each page often cannot be used. After you incorporate padding and possible rowID to ascertain the "real row length", you can calculate unused space per-page as follows:

  ```
  65536 - real_rowlen * FLOOR(65536 / real_rowlen)
  ```

- multi-pass merge: doubling SORT procedure utility files (or sort failure)

  Sorts of extremely large data might cause a multi-pass merge. The wider the uncompressed data (padded row width that is fed to the sort) and the more rows, the higher the chance of a multi-pass merge. A value for `SORTSIZE=` that is too small can also cause a multi-pass merge.

  The merge phase combines procedure utility file runs and completes the sort. Usually, a merge is single-pass. No additional space consumption or other considerations are needed for a single-pass merge.

  A multi-pass merge generates a copy of every row that is in the procedure utility file, which doubles space consumption. In addition, a multi-pass merge is not yet implemented in a SAS®9 multi-threaded sort; the rare multi-pass condition fails due to insufficient memory. There are two work-arounds for a SAS®9 multi-pass merge condition:

  - increase the value for `SORTSIZE=`
  - request a SAS 8 sort and use the `NOTHREADS` option.

  The following simple equation estimates the number of rows that you can sort in a single-pass merge. A number of rows close to or exceeding this figure might require a multi-pass merge.

  ```
  (SORTSIZE/64K) * (SORTSIZE/row_width)
  ```

  In the preceding equation:

  - 64K is the default I/O page size of the SORT procedure utility file. This can be changed for the SORT procedure but not for joins.
  - row_width is the length of each row that is passed for sorting, including pad bytes.
  - SORTSIZE/64K is the maximum number of procedure utility file runs that are possible for a single-pass merge.
  - SORTSIZE/row_width is the number of rows per run, that is, the number of rows that can be sorted in memory constrained by the value of `SORTSIZE=`.

  This equation shows the number of rows that handled single-pass increases with smaller row width and a larger value for `SORTSIZE=`.

  The approximate figure is the theoretical maximum; in reality, fewer rows can be handled before a multi-pass condition. The merge phase of a SAS®9 multi-threaded sort requires a 64K-memory allocation for each run. Each 64K-allocation that is run is charged against the `SORTSIZE=` memory limitation. Assuming `SORTSIZE=`512M and ignoring overhead for data structures, the limit is 8192 runs (512M / 64K). That is, the maximum number of runs that are processed during a single-pass when `SORTSIZE=`512M is about 8000. Ignoring overhead, the number of rows per run is SORTSIZE / row_width, where row_width includes pad bytes.

A simple example with an extremely small SORTSIZE of 256K further illustrates this idea. Only several runs are accommodated in a single-pass. With a row length of almost 32K (32000+8), a run only holds 8 rows. The text with the examples explains their progression. In the following example, the `THREADS` option forces you to use a SAS®9 multi-threaded sort. The `DETAILS` option outputs SORT procedure utility file information.

With 8 rows, the sort completes in memory, because 256K / 32K = 8.

```
%let numrows=8;

data a; length chr $32000; length i 8.;

do i=1 to &numrows; output; end; run;


option sortsize=256k;

proc sort threads details data=a out=b; by i; run;


NOTE: There were 8 observations read from the data set WORK.A.

NOTE: Sort completed in memory.
```

There are 3 runs (24 / 8). The sort completes because
(3 * 64K) + overhead < 256K.

```
%let numrows=24;

data a; length chr $32000;

do i=1 to &numrows; output; end; run;


option sortsize=256k;

proc sort threads details data=a out=b; by i; run;


NOTE: There were 24 observations read from the data set WORK.A.

NOTE: 3 sorted runs written to utility file.

NOTE: Utility file contains 12 pages of size 65536 bytes for a
      total of 768.00 KB.
```

There are 4 runs (32 / 8). The sort fails because (4 * 64K) + overhead > 256K, which forces a multi-pass sort.

```
%let numrows=32;

data a; length chr $32000;

do i=1 to &numrows; output; end; run;


option sortsize=256k;

proc sort threads details data=a out=b; by i; run;


NOTE: There were 32 observations read from the data set WORK.A.

NOTE: 4 sorted runs written to utility file.

NOTE: Utility file contains 16 pages of size 65536 bytes for a
      total of 1024.00 KB.

ERROR: Insufficient memory.
```

Note: Increasing the sort size might not always help the performance of your sort. The amount of memory that you have, the number of rows, and the width of the rows all affect sorting performance.

Sizing Output Data

To size the output data, apply the sizing rules of the destination data store to the columns that are produced by the sort. For our purposes, we assume output to a SAS data set. A SAS output data set eliminates most padding. The data set physically arranges 8-byte, non-character columns first, followed by short numeric values and character columns packed together. The only padding is at the end of a row, which forces the next row to have an 8-byte boundary. Other SAS data set rules apply to predict space for the output data. For example, compression usually reduces space consumption. For more information, see "Indexing and Compressing SAS® Data Sets: How, Why, and Why Not" (Karp 1995) at www2.sas.com/proceedings/sugi25/25/aa/25p005.pdf.

Note: If your input and output data are in the same SAS data set, you can decrease the maximum disk space that is required by using the `OVERWRITE` option. However, be aware that you are using the `OVERWRITE` option because an overwrite deletes the input data set before writing the same named output data set with the sorted data. It is important to notice if an I/O error or out-of-space error appears during output because this can cause loss of critical data.

Sizing output relational database data should not be an issue when you use PROC SORT. Relational databases do not maintain data in sorted order.

SAS SQL Joins

Joins are a common and resource-intensive part of ETL. SAS SQL implements several well-known join algorithms: sort-merge, index, and hash. There are common techniques to aid join performance, irrespective of the algorithm that is chosen. Conditions often cause the SAS SQL optimizer to choose the sort-merge algorithm; techniques that improve sort performance also improve sort-merge join performance. However, understanding and leveraging index and hash joins will result in performance gains.

It is common in ETL to perform lookups between tables, especially when building Star Schemas. Based on key values in one table, you look up matching keys in a second table and retrieve associated data in the second table. SQL joins can perform lookups. However, SAS and SAS Data Integration Studio provide special look-up mechanisms that usually outperform a join. For details, see the section "Lookups in a Star Schema".

Here are some misconceptions that are associated with joins:

- Join performance is slow.
- It's not clear how joins, especially multi-way joins, work under the "covers".
- There is no way to influence which join algorithm SAS SQL chooses.
- The amount of disk space that is needed to complete the join, especially multi-way joins, is very large
- How SAS SQL joins work when joining relational database data and SAS data sets is not optimal.

The next sections address these misconceptions.

Optimizing Join Performance

If you understand the attributes of your tables such as table size, join keys, column cardinality, and so on, you can use this information to help you optimize the SAS SQL join strategy. Keep in mind that, over time, table attributes might change, and the join strategy might need to be adapted. The following best practices should help you optimize performance of the joins.

Pre-Sorting Data for Joins

Pre-sorting can be the most effective means to improve overall join performance. Usually, a table that participates in multiple joins on the same join key benefits from pre-sorting. A word of caution here. In a pre-sort, all retained rows and columns in the table are "carried over" during the sort; the SORT procedure utility file and output data set are relatively large. Pre-sorting might not be needed if you have large amounts of data, and you are subsetting the data for the join.

Optimizing Order of Columns

Columns should be ordered in the SELECT statement as follows:

- non-character (numeric, date, datetime) columns first
- character columns last.

In a join context, all non-character SAS columns consume 8 bytes. This includes SAS short numeric columns. Ordering non-character (numeric, date, datetime) columns first, perfectly aligns internal SAS SQL buffers so that the character columns, which are listed last in the row returned by the execution of the SELECT statement are not padded. The result is fewer I/Os and lower CPU consumption, particularly in the sort phase of a sort-merge join.

Tip: In SAS Data Integration Studio, use a transformation's **`Mapping`** tab to re-arrange columns by column type. Do this by sorting the "type" column in the target table in descending order.

Dropping Unneeded Columns

Joins tend to be I/O intensive. To help minimize I/O and improve performance, it is recommended that you drop unneeded columns, minimize column widths (especially from relational database tables if they have wide columns), and delay the expansion of column widths (this usually happens when you combine multiple columns into a single column to use a key value) until the end of the ETL flow. See the tips in the section "Transformation and Column Optimizations"

Options to Influence SQL Selection of Join Algorithm

Options exist to influence SAS SQL to use a particular join algorithm. Use these options to influence SAS SQL from using the join algorithm that is selected by the optimizer to the join algorithm that you prefer.

Note that these options can only influence the optimizer; they cannot force it. If the optimizer cannot perform the type of join that you have selected, then it will select the type of join that it considers best for the data.

Before discussing the different join algorithms, you need to learn how to determine which join algorithm your SAS SQL code is using. To do this, specify the add _METHOD parameter in the PROC SQL statement. This parameter writes debugging trace output in the SAS log file.

Also, it is important to understand the keywords that are used in the trace output.

```
sqxsort:   sort step

sqxjm:     sort-merge join

sqxjndx:   index join

sqxjhsh:   hash join

sqxsrc:    table name
```

The following fragment examples show these keywords in a _METHOD trace.

In this example, each data set is sorted, and sort-merge is used for the join.

```
sqxjm
    sqxsort
        sqxsrc( WORK.JOIN_DATA2 )
    sqxsort
        sqxsrc( LOCAL.MYDATA )
```

In this example, an index nested loop is used for the join.

```
sqxjndx
    sqxsrc( WORK.JOIN_DATA2 )
    sqxsrc( LOCAL.MYDATA )
```

In this example, a hash is used for the join.

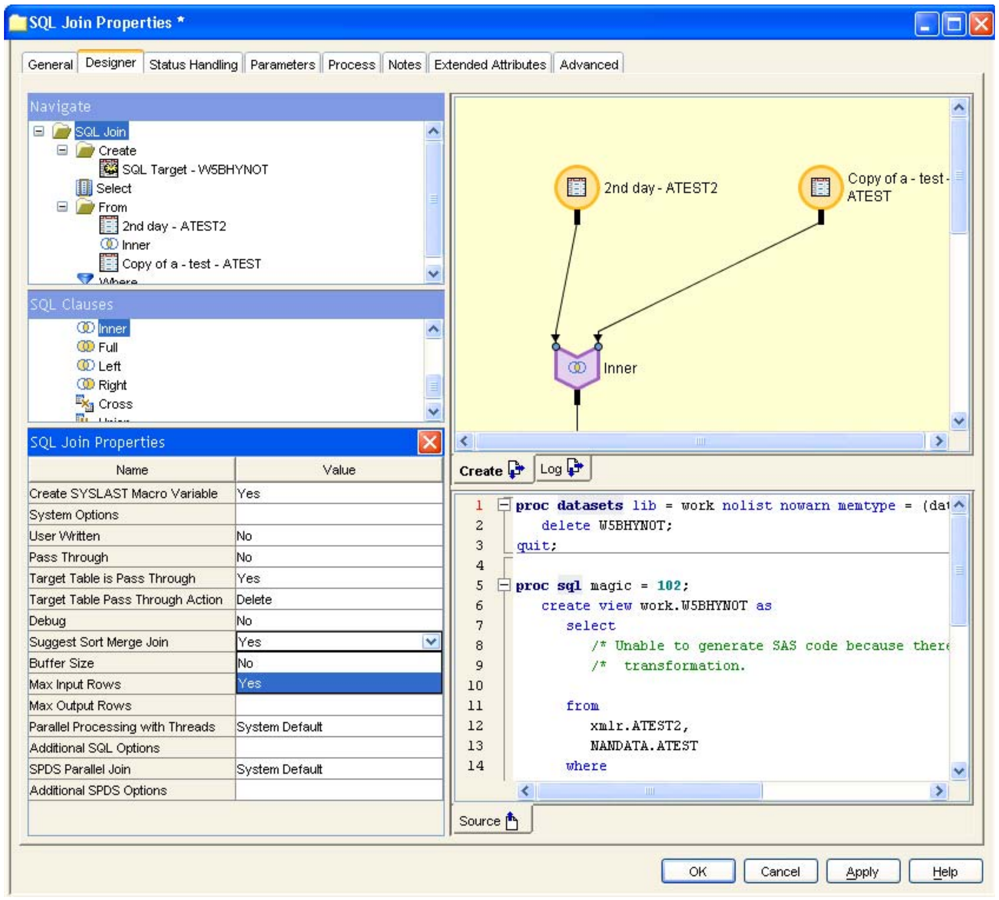
```
sqxjhsh
                  sqxsrc( LOCAL.MYDATA )
                  sqxsrc( WORK.JOIN_DATA1 )
```

Now, let’s examine the types of SAS SQL joins.

Sort-Merge Joins

Sort-merge is the algorithm most often selected by the SQL optimizer. As the name implies, the two files that are being joined are sorted and then merged. Sorting is the most resource-intensive aspect of a sort-merge join. Therefore, techniques that improve sort performance do the most to improve join performance. To optimize sorting, see the section "Sorting Data", which contains a detailed discussion about what happens behind-the-scenes in a sort. Briefly, intermediate files are placed in the SAS WORK location unless the `-UTILLOC` invocation option is specified. For very large joins, using `-UTILLOC` to segregate I/O activity is recommended. For more information, see the section "Sorting Data".

To encourage a sort-merge join algorithm in SAS Data Integration Studio, select the **Suggest Sort Merge Join** property in the lower-left panel in the SQL Join Properties window of the SQL Join transformation (Figure 6) to add `MAGIC=102` to the PROC SQL invocation.



SQL Join Properties

Name	Value
Create SYSLAST Macro Variable	Yes
System Options	
User Written	No
Pass Through	No
Target Table is Pass Through	Yes
Target Table Pass Through Action	Delete
Debug	No
Suggest Sort Merge Join	Yes
Buffer Size	No
Max Input Rows	Yes
Max Output Rows	
Parallel Processing with Threads	System Default
Additional SQL Options	
SPDS Parallel Join	System Default
Additional SPDS Options	

```
proc datasets lib = work nolist nowarn memtype = (dat
   delete W5BHYNOT;
quit;

proc sql magic = 102;
   create view work.W5BHYNOT as
      select
         /* Unable to generate SAS code because there
         /*  transformation.

      from
         xmlr.ATEST2,
         NANDATA.ATEST
      where
```

Figure 6. Sort-Merge Join Property Turned On in Version 2 of SQL Join Transformation

You can also directly code the options that are added in SAS Data Integration Studio by adding `MAGIC=102` to the PROC SQL invocation,as follows:

```
proc sql _method magic=102;
```

Caution: If you code the `MAGIC=` option directly into your SAS SQL, be aware that this option is subject to change in future releases of SAS, and a change might cause your code to become invalid. If you deploy the `MAGIC=` option in SAS 9.1.3 or earlier, be prepared to change your code.

Index Joins

An **index join** looks up each row of the smaller table by querying an index of the large table. When chosen by the optimizer, an index join usually outperforms a sort-merge on the same data. The SAS SQL optimizer considers an index join under the following conditions:

- The join is an equijoin—tables are related by equivalence conditions on key columns.
- There are multiple conditions, and they are connected by the AND operator.
- The larger table has an index composed of all the join keys.

Example 1: Eligibility for Index Join

A single equijoin condition on column ASSESSMENT_RATING_GRADE_RK:

```
select COUNTERPARTY.ASSESSMENT_MODEL_RK,
       COUNTERPARTY.ASSESSMENT_RATING_RK,
       ASSESSMENT.CCF_MAX_ASSESSMENT_RT
from DDS.COUNTERPARTY INNER JOIN CRMART.ASSESSMENT
 ON (COUNTERPARTY.ASSESSMENT_RATING_RK = ASSESSMENT.ASSESSMENT_RATING_RK);
```

Assuming an ASSESSMENT on the larger table and indexed on ASSESSMENT_RATING_RK, the join is eligible for an index join.

A composite index on (ASSESSMENT_RATING_RK, ASSESSMENT_MODEL_RK) does not qualify for an index join in Example 1.

Example 2: Eligibility for Index Join

An equijoin on ASSESSMENT_RATING_GRADE_RK AND ASSESSMENT_MODEL_RK:

```
select COUNTERPARTY.ASSESSMENT_MODEL_RK,
       COUNTERPARTY.ASSESSMENT_RATING_RK,
       ASSESSMENT.CCF_MAX_ASSESSMENT_RT
from DDS.COUNTERPARTY INNER JOIN CRMART.ASSESSMENT
ON (COUNTERPARTY.ASSESSMENT_RATING_RK = ASSESSMENT.ASSESSMENT_RATING_RK
    AND
    COUNTERPARTY.ASSESSMENT_MODEL_RK = ASSESSMENT.ASSESSMENT_MODEL_RK);
```

Assuming an ASSESSMENT on the larger table and a composite index on either (ASSESSMENT_RATING_RK, ASSESSMENT_MODEL_RK) or (ASSESSMENT_MODEL_RK,ASSESSMENT_RATING_RK), the join is eligible for an index join.

A simple index on ASSESSMENT_RATING_RK does not qualify for an index join in Example 2.

After a table is assessed as eligible for an index join, the optimizer factors the relative table sizes and key cardinality to decide whether to perform an index join. The optimizer assumes that every row in the small table has one or more matching values in the large table. Further, the optimizer uses the cardinality of the large table to predict how many matches occur per small table row.

For example, assume that the small table has 10 rows and the large table has 1,000,000 rows with 500,000 distinct values on an indexed equijoin key. The optimizer predicts 10 * (1,000,000 / 500,000) = 20 matches. Then, if the predicted number of matches is less than 15% of the rows in the large table, SAS SQL uses the index. In this case, the predicted number is 20 < 150,000, and an index join occurs.

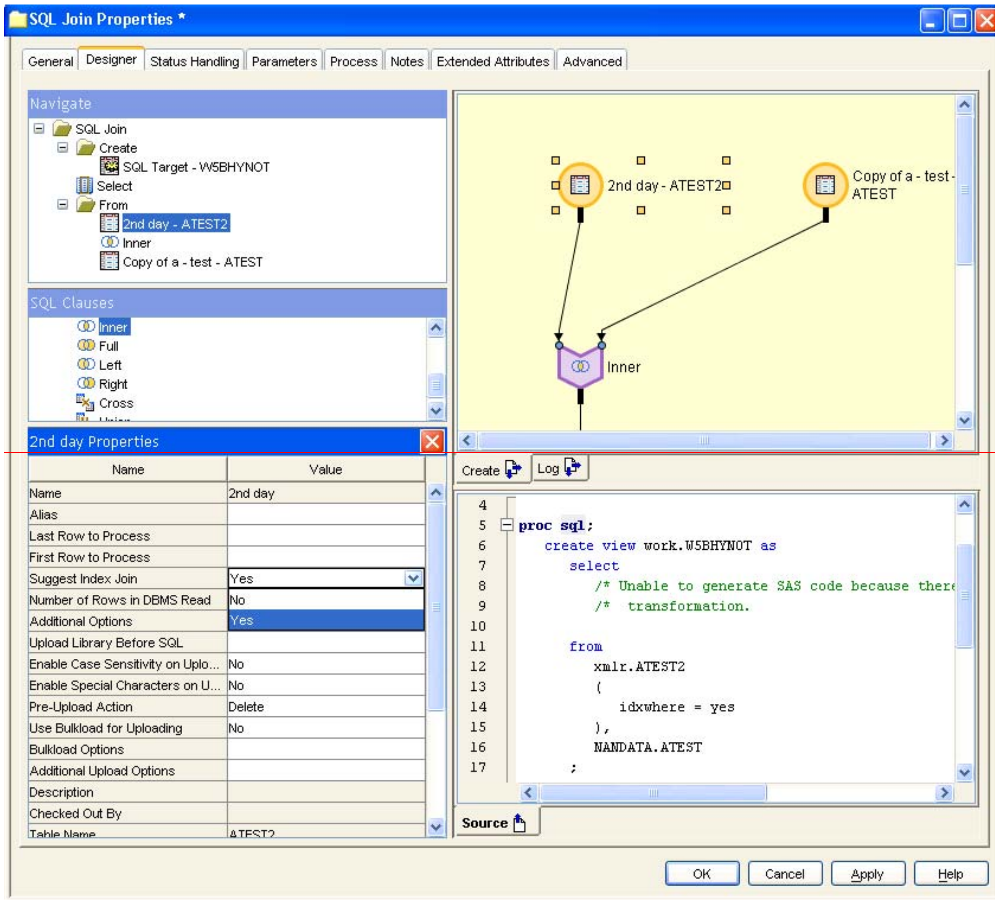
Option to Influence Index Join Algorithm

Usually, an index join outperforms a sort-merge join on the same data. However, the optimizer cost equation for choosing the index is conservative. If you suspect that the optimizer equation is overly conservative for your particular data, encourage an index join with the data set option `IDXWHERE=YES`. You might want to influence the optomizer to use an index join if the following conditions exist:

- There are relatively few matches on the join key. Although the optimizer uses a prediction of 15% matching rows in the large table to decide on an index join, you might find that an index join outperforms the sort-merge join when the the table contains up to 25% actual matches.
- The tables have very wide row widths, which decreases performance in a sort-merge join. "Very wide" refers to the decompressed width.
- An indexed large table is partially ordered on the join key, which lowers I/O for indexed access.
- A small (sequentially scanned) table is ordered on the join key, which optimizes index "probes" on the larger table.

There must be an appropriate index on the larger table; otherwise the option `IDXWHERE=YES` has no effect. For additional information about indexes and index joins, see the section "Indexes on SAS Data Sets".

Alternatively, when using SAS Data Integration Studio, turn on the **Suggest Index Join** property in the 2nd day Properties panel in the lower-left corner of the SQL Join Properties window for an input table in the SQL Join transformation of SAS Data Integration Studio (Figure 7).



2nd day Properties

Name	Value
Name	2nd day
Alias	
Last Row to Process	
First Row to Process	
Suggest Index Join	Yes
Number of Rows in DBMS Read	No
Additional Options	Yes
Upload Library Before SQL	
Enable Case Sensitivity on Uplo...	No
Enable Special Characters on U...	No
Pre-Upload Action	Delete
Use Bulkload for Uploading	No
Bulkload Options	
Additional Upload Options	
Description	
Checked Out By	
Table Name	ATEST2

```

proc sql;
   create view work.W5BHYNOT as
      select
         /* Unable to generate SAS code because there
         /*  transformation.

      from
         xmlr.ATEST2
         (
            idxwhere = yes
         ),
         NANDATA.ATEST
      ;
```

Figure 7. Suggest Index Join Property Turned On in Version 2 of the SQL Join Transformation

Hash Joins

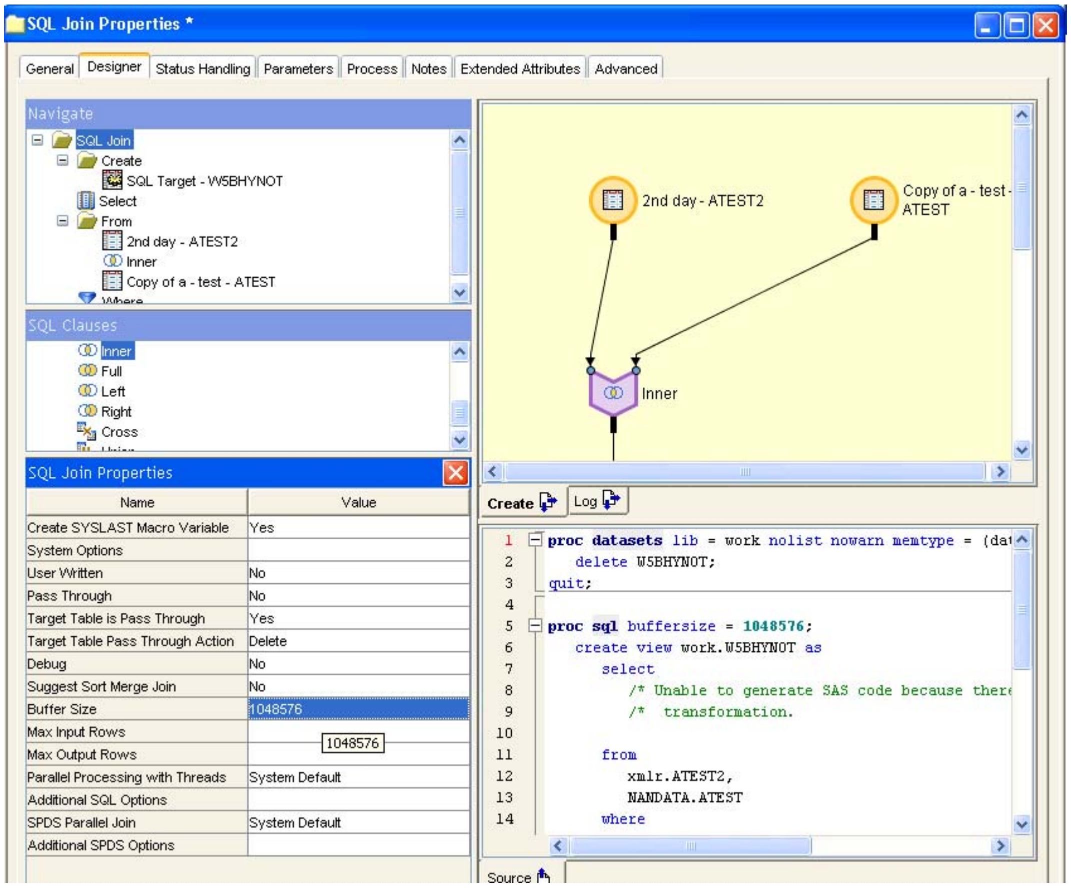
With a hash join, the smaller table is re-configured in memory as a hash table. SAS SQL sequentially scans the larger table and, row-by-row, performs hash lookup against the small table to create the result set.

A memory-sizing formula, which is not presented here, determines if a hash join is chosen. The formula is based on the PROC SQL option `BUFFERSIZE=`, whose default value is 64K. Especially on a memory-rich system, consider increasing the value for `BUFFERSIZE=` in order to increase the likelihood that a hash join is chosen.

Option to Influence Hash Join Algorithm

Encourage a hash join by increasing the default value of 64K for the `BUFFERSIZE=` option in PROC SQL, as shown in Figure 8. Here is the PROC SQL statement for this.

```
   proc sql _method buffersize= 1048576;
```



SQL Join Properties

Name	Value
Create SYSLAST Macro Variable	Yes
System Options	
User Written	No
Pass Through	No
Target Table is Pass Through	Yes
Target Table Pass Through Action	Delete
Debug	No
Suggest Sort Merge Join	No
Buffer Size	1048576
Max Input Rows	
Max Output Rows	
Parallel Processing with Threads	System Default
Additional SQL Options	
SPDS Parallel Join	System Default
Additional SPDS Options	

```
proc datasets lib = work nolist nowarn memtype = (dat
   delete W5BHYNOT;
quit;

proc sql buffersize = 1048576;
   create view work.W5BHYNOT as
      select
         /* Unable to generate SAS code because there
         /*  transformation.

      from
         xmlr.ATEST2,
         NANDATA.ATEST
      where
```

Figure 8. Changing Buffer Size Property in Version 2 of the SQL Join Transformation

Multi-Way Joins

Many joins are two-table joins. Multi-way joins can join more than two tables and are common in Star Schema processing.

In SAS SQL, a multi-way join is executed as a series of joins between two tables. The first two tables that are joined create a temporary result table. That result table is joined with a third table from the original join, which results in a second result table being created. This pattern continues until all the tables in the multi-way join have been processed, and the final answer set is created. For example, consider a multi-way join on tables TAB1, TAB2, TAB3, and TAB4. Here is how the process might be handled by SAS SQL:

- join TAB2 and TAB3 to create temporary result table TEMP1
- join TEMP1 and TAB4 to create temporary result table TEMP2
- join TEMP2 and TAB1 to create the final result.

Note: Prior to SAS 9.1.3 Service Pack 4, SAS SQL joins were limited to 32 table references. For SAS 9.1.3 Service Pack 4 and later, the SAS SQL join limit has been increased to 256 table references. The SQL procedure can internally reference the same table more than once, and each reference counts as one of the allowed table references.

Disc Space Consumption of Multi-Way Joins

SAS SQL does not release temporary tables, which reside in the SAS WORK directory, until the final result is produced. Therefore, the disk space that is needed to complete a join increases with the number of tables that participate in a multi-way join.

Whether they reside in SAS WORK or in UTILLOC, the sort utility files are deleted when an intermediate sort-merge join completes. This means that, unlike temporary result tables, sort utility files are not retained throughout a multi-way join.

Order of Processing in Multi-Way Joins

By nature, the SAS SQL inner join operation allows the SAS SQL optimizer to re-arrange the join order. An inner join of tables A, B, and C results in the same answer if the operation is performed in either of the following ways:

```
((A inner join B) inner join C)
```

or

```
((B inner join C) inner join A)
```

The join syntax that you use for inner joins determines if the SAS SQL optimizer will attempt to re-arrange inner joins. The recommended syntax is to delimit inner join table references with a comma, and place the join condition in the WHERE clause. When you use the recommended syntax, you enable SAS SQL to apply cost analysis and to re-arrange the ordering of inner joins. In general, SAS SQL chooses the optimal order.

The alternate inner join syntax spells out INNER JOIN between table references and places the join condition(s) in the ON clause. This syntax disables re-ordering of inner joins. If you determine that the SAS SQL optimizer makes a non-optimal choice in ordering the inner joins that are specified with the recommended syntax, you can explore the alternate syntax as a means of optimizing performance.

Join Algorithms in Multi-Way Joins

The optimizer re-orders execution to favor index usage on the first join that is executed. Subsequent joins do not use an index unless encouraged with the `IDXWHERE` option. Based on row counts and the `BUFFERSIZE=` value, subsequent joins are executed with a hash join if they meet the optimizer formula. In all cases, a sort-merge join is used when neither an index join nor a hash join are appropriate.

Relational Database Considerations

There are two types of joins that can occur between relational database tables and SAS data sets:

- **homogenous joins** in which all the tables reside in a single schema of the relational database
- **heterogeneous joins** in which relational database table(s) are joined with SAS data set(s).

In homogenous joins, SAS SQL attempts to push the SQL code that is associated with the join to the realational database. Alternatively, if some of the tables to be joined are from a single database instance, SAS SQL attempts to push the joins that reference data tables to the database. Any joins that are performed by a database are executed with database-specific join algorithms; the rules and algorithms of SAS do not apply.

In heterogeneous joins, SAS SQL first tries to push any WHERE-clause processing to the database to handle. Then SAS SQL pulls all the data from the relational database table into SAS and joins the heterogeneous data in SAS. Performance problems can sometimes occur in heterogeneous joins. This occurs when SAS SQL chooses a hash join. The following example shows that influencing SAS SQL to use a sort-merge join by using `MAGIC=102` can perform the join faster.

```
proc sql _method; create table new as select * from
join_data1,orcle.join_data2

where join_data1.var1 = join_data2.var1;
```

SQL trace:

```
sqxcrta

    sqxjhsh

        sqxsrc( WORK.JOIN_DATA1 )

        sqxsrc( ORCLE.JOIN_DATA2 )
```

```
proc sql _method magic=102; create table new as select * from
join_data1,orcle.join_data2

where join_data1.var1 = join_data2.var1;
```

SQL trace:

```
sqxcrta

    sqxjm

        sqxsort

            sqxsrc( WORK.JOIN_DATA1 )

        sqxsort

            sqxsrc(ORCLE. JOIN_DATA2)
```

Star Schema Optimization

The SAS SQL optimizer has no dedicated logic for Star Schema optimization.

Indexes on SAS Data Sets

In the context of large-scale ETL flows, indexes on SAS data sets might be useful to perform the following processes:

- resolve WHERE clauses
- allow an index join as an optimization for two-table equijoins
- verify key uniqueness.

WHERE Clauses Resolved in ETL Flows

In the context of ETL flows, indexes on SAS data sets are beneficial for processing selective WHERE clauses that qualify for index use. When resolving a WHERE clause, if the number of rows that qualify for the subsetting is 15% or less than the total number of rows, then the index is used; otherwise, a sequential scan occurs. Keep this in mind when you build indexes because the overhead of building and maintaining an index is costly.

Index selection for resolving WHERE clauses and examples are contained in the following paper: "Indexing and Compressing SAS Data Sets: How, Why and Why Not", (Karp and Shamlin 1993) available at www2.sas.com/proceedings/sugi28/003-28.pdf.

Allowing Index Joins

As described in the earlier section "Index Joins", if several conditions are met (such as, equijoin, one table much smaller, larger table indexed, high key cardinality in larger table), SAS SQL performs an index join.

Only equijoins are eligible for an index join. If there are multiple join conditions, they must be connected by the AND operator.

Index use is limited in multi-way joins, which leverage only one index. Multi-way joins are solved as a series of two-way joins. The SAS SQL optimizer samples keys and, based on the sample distribution, arranges the join order to optimize overall execution time. The optimizer favors an index join for the first two tables joined, but a usable index must exist. Sampled distributions favor an index join. On subsequent sub-joins, SAS SQL does not use an index.

Here is an example. Three tables have an index on column **i** and an index on **randvar**. The _METHOD trace output shows that an index join was used in the first sub-join (WORK.JOIN_DATA2 joined to LOCAL.COMPRESS_90PERCENT). However, a sort-merge is used for the next sub-join. An index join is represented by sqxjndx. A sort-merge is represented by sqxjm.

```
proc sql _method;

create table _null as select * from
join_data1,join_data2,local.compress_90percent

where join_data1.randvar = compress_90percent.randvar

and join_data2.i = compress_90percent.i;

quit;
```

```
      sqxcrta

          sqxjm

              sqxsort

                  sqxsrc( WORK.JOIN_DATA1 )

              sqxsort

                  sqxjndx

                      sqxsrc( WORK.JOIN_DATA2 )

                      sqxsrc( LOCAL.COMPRESS_90PERCENT )
```

An index join exhibits best performance when there is a high percentage of non-matches between the join tables. In this circumstance, most attempted matches "short-circuit" because the key is not present in the indexed table. That is, for non-matches, the join accesses only the index of the large table, not the table itself. The optimizer that chooses an index join usually provides superior performance results when compared to a sort-merge join, but superior performance is not guaranteed.

For a comparison of index joins to other SAS SQL join algorithms, see the section "SAS SQL Joins".

Verifying Key Uniqueness

With the UNIQUE qualifier, an index build verifies key uniqueness. Here is an example:

```
proc sql; create unique index Cust_ID on mydata.customer; quit;
```

Index Used in SAS?

To determine if SAS uses an index, use the SAS option `MSGLEVEL=`:

```
option msglevel=i;
```

For information about index usage by SAS SQL, use the `_METHOD` trace option:

```
proc sql _method;
```

Encouraging Index Usage

In general, let SAS determine if an index should be used for WHERE clauses and joins. To encourage SAS to use an index with a specific SAS data set, use the option `IDXWHERE=YES`. However, be aware that encouraging index usage can degrade performance. PROC SQL applies intelligence in declining to use an index.

Indexes for Sorting

Indexes are not used by SAS for sorting.

Environment Tuning

Each ETL flow interacts with the hardware and the operating environment that it runs on to accomplish its tasks. In addition to ETL code and underlying methods and options, actual performance is highly dependent on correctly setting up the hardware environment for maximum capacity and throughput. Appropriate resource allocation for each concurrent process ensures good performance.

The following hardware and software tuning recommendations and comments are based on experience with the SAS ETL flows. While developing your ETL flows, you should determine the amount of disk space that is required for the various SAS libraries that are used by the ETL flow (the size of the SAS WORK area is the hardest to determine) if you want the ETL flow to complete in a timely fashion. After you know the size of the various libraries, you can determine how to balance the I/O subsystems to ensure that you have the best I/O throughput possible for the computer that you are running. There is a lot of discussion in this paper about tuning the I/O, because experience has shown that I/O bottlenecks are the leading reason for poor performance of ETL flows. A traditional ETL flow is usually a single-threaded process, so achieving your desired service levels might require breaking a single ETL flow into multiple flows that can execute concurrently. Periodically, you should monitor the ETL flow and the hardware it is running on in order to make sure that there are no bottlenecks so that the job will run in the required time frame.

Disk Space Maximum for Intermediate Files

As described in the section "Intermediate Files for SAS Data Integration Studio Jobs", SAS Data Integration Studio jobs create many intermediate files. There have been situations in which ETL flows fail because the flows completely fill up the directory that is used for SAS WORK. Determining the maximum amount of space that is required for intermediate files is valuable when you are configuring the computer to support your ETL flows. Configuring the amount of WORK cannot be done by turning on a SAS parameter, but there are ways outside of SAS to track the disk space that is used by intermediate files.

If you direct the UTILLOC directory(s) to a location other than SAS WORK, or you are heavily using another SAS library, run additional scripts to capture the maximum amount of disk space for the additional directory(s).

Disk Space Usage for Intermediate Files

Managing intermediate files throughout the ETL flow can save valuable disk space, especially if the amount of disk space that is available to the ETL flow is restricted.

As described in the section "Deleting Intermediate Files", intermediate files are usually deleted after they have served their purpose. However, it is possible that some intermediate files might be retained longer than desired in a particular ETL flow. For example, some user-written transformations might not delete the temporary files that they create, and scheduled ETL flows to the batch server might run out of disk space for WORK files.

The following post-processing macro can be incorporated into an ETL flow. This macro uses the DATASETS procedure to delete all data sets in the WORK library, including any intermediate files that have been saved to the WORK library.

```
%macro clear_work;

    %local work_members;

    proc sql noprint;

    select memname

    into :work_members separated by ","

    from dictionary.tables

    where

           libname = "WORK" and

           memtype = "DATA";

    quit;

    data _null_;

           work_members = symget("work_members");

           num_members = input(symget("sqlobs"), best.);

           do n = 1 to num_members;

                  this_member = scan(work_members, n, ",");

                  call
symput("member"||trim(left(put(n,best.))),trim(this_member));

           end;

           call symput("num_members",
trim(left(put(num_members,best.))));

    run;

    %if &num_members gt 0 %then %do;

           proc datasets library = work nolist;

                  %do n=1 %to &num_members;

                         delete &&member&n;

                  %end;

           quit;

    %end;

%mend clear_work;

%clear_work
```

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

Deleting Transformation Output Tables in a Long ETL Flow

The transformation output tables for an ETL flow remain until the SAS session that is associated with the flow is terminated. As you analyze what is happening behind-the-scenes with your ETL flow, and you determine that there are many output tables that are not being used, you might want to add nodes to the ETL flow that will delete these output tables (especially, if these tables are large). Deleting the output tables will free valuable disk space and memory.

For example, you can add a **generated transformation** that would delete output tables at a specified point in the flow. For details about generated transformations, see the section "Add Your Own Transformation to the Process Library".

Hardware Considerations for ETL Flows

As mentioned in the introduction to this section, I/O bottlenecks are the leading causes of ETL flows not finishing within the desired time frame. Here is some information about what you should look for when you set up the file systems on your computer, and some general information about what to look for with regard to your ETL flow. From a hardware perspective, the best way to improve I/O is to ensure the following items:

- adequate I/O bandwidth for ETL flows
- sufficient disk spindles (to ensure the I/O throughput rate that is required to achieve the best I/O performance)
- fastest I/O rate is available to the various SAS libraries (from a hardware perspective, do this by striping the file system across multiple disks and I/O channels).

It is best to work with the IT administrators at your site in order to determine how to best configure the hardware, based on SAS usage. Knowing the amount of I/O that occurs during each transformation and how quickly the transformation must be complete, will help your IT administrators determine the I/O throughput rate that is required to ensure that a transformation finishes within the desired time frame. Additional resources to help you with hardware configuration issues are: the SAS Support Consultant at your site, a Technical Support Consultant at SAS, a partner vendor, or the *SAS 9.1.3 Intelligence Platform Planning and Administration Guide* available at support.sas.com/documentation/configuration/iaplanning913.pdf.

SAS ETL flows benefit when multiple file systems are configured with many, independent disks across independent I/O channels or fast I/O channels. Smaller disks have more spindles. More spindles result in faster I/O throughput rates (a minimum of 200 MB/sec up to 800 MB/sec), which makes this type of configuration ideal for the heavily-used SAS WORK and UTILLOC libraries, and effectively distributes I/O and boosts performance.

Understanding the SAS code that is generated by your SAS Data Integration Studio flow is critical to effectively deploying available volumes. If you understand the code that is generated, you can determine if a faster I/O throughput rate is needed because of the sequential access of the data. If there will be more random access to the data, you can determine if a high number of IOPs are required. These determinations can also help your system administrator to decide the best way to set up the I/O sub-systems.

File Cache Management

By default, SAS asks the operating environment to retrieve data from and write data to the various file systems on the computer. The operating environment performs these tasks by reading the data from disk into file cache (that is, memory on the computer). One of the recent improvements in operating environments results in reads that are more efficient. The operating environment analyzes how the application is requesting the data and initiates activities on behalf of the application. With SAS performing large sequential reads, the file system becomes very aggressive in performing read-aheads of the data so that the data is in memory and ready to be accessed by SAS. This is good until it causes the computer to **thrash** physical memory that is used by the SAS application and memory that is used by the file cache. When the computer begins to swap real memory pages out to disk (for example, virtual memory), then performance is degraded. For these reasons, ensure that there is enough memory to support SAS activities, file cache, and other operating environment requirements. It is essential that you monitor the memory usage to make sure that there is no thrashing (on UNIX, the lru and sync daemons begin accruing a lot of time) on the server. If this occurs, then you should restrict the amount of memory used by file cache (this can be done on most of the UNIX operating environments), or you should restrict the amount of memory that SAS can access.

Note: If the amount of memory for file cache is restricted too much, the performance of SAS is degraded because too great a restriction of file-cache memory disables its read-ahead capability.

I/O Performance Improvement

When you set up file systems, consider enabling read-ahead, write-behind, and file cache free-behind; disable write-throttling; and look for a lack of physical I/O buffers as reported by the operating environment tools. These features are not available on all file systems or on all operating environments, and the actual settings for the file system are based on the actual I/O access patterns of the ETL flow.

Consider balancing the I/O requests from SAS (via the `BUFSIZE=` option) with the I/O buffers of the underlying I/O sub-system. If you can set the SAS I/O buffer and the operating environment I/O buffer to be the same size or to be a multiple of one another (setting the operating environment buffer larger than the SAS buffer), you can streamline the process of getting data into SAS and the processing might be faster.

Here are some recommendations for improving I/O performance under various operating environments:

- Under z/OS, when you specify sequential access of files, choose a library block size that corresponds to half-track blocking, that is, two blocks per track. For example, specify:

```
option blksize(3380)=half blksize(3390)=half;
```

- Under z/OS, when you specify random access of files, choose a library block size of 6K, if that block size is practical. For some DASD controller configurations, half-track blocking performs almost as well as a 6K block size for random access. Half-track blocking, which results in fewer inter-block gaps, allows more data to be packed on a track.
- Under Windows and UNIX, it is recommended that `BUFSIZE=` be set to 64K when you are executing large, sequential Reads and Writes. You might want to increase the value of `BUFSIZE=` to 128K if you have a fast I/O sub-system. However, a value higher than 128K for this option is not recommended.

Occasionally, SAS is asked about setting up in-memory file systems to improve I/O performance. Depending on the application, using in-memory file systems can improve performance by having the data close to the processor. However, as soon as you need more space than you have physical RAM, swapping begins and you will see a big degradation of performance. Using in-memory file systems has been used, successfully, when an application uses a single file (or several small files) that can fit in the memory on the computer. Putting that file into an in-memory file system might improve performance:

- Under z/OS, consider using the In-Memory File (IMF) feature for a SAS file that will be accessed across many SAS steps (DATA steps or procedures) if the file is small enough to fit in the available region size. Load the file by specifying the SASFILE statement before specifying the SAS steps that will process the file. The file will be read and, if necessary, written only once. Without IMF, the file would be read once per step. For more information about the SASFILE statement, see the "SAS Companion for z/OS" under Base SAS, Operating Environment Specific Information in *SAS OnlineDoc® 9.1.3* available at support.sas.com/onlinedoc/913/docMainpage.jsp.
- Under Windows, use the `MEMLIB` and `MEMCACHE` options to create memory-based SAS libraries. Depending on your operating environment, extended memory or conventional memory supports these memory-based libraries. For more information, see "SAS Language Reference: Dictionary" under Base SAS in *SAS OnlineDoc® 9.1.3* available at support.sas.com/onlinedoc/913/docMainpage.jsp.
- Under UNIX, many of the operating systems can create in-memory file systems. Consult your System Administrator to learn which operating systems are available.

Direct I/O

An alternative to restricting memory is to have SAS execute direct I/O (that is, directly access data from disks), where possible, and not use file cache. This alternative process decreases the amount of memory that is used by the ETL flow because the SAS process accesses the data directly from disk and not from memory (file cache). This alternative process is best used in the following situations:

- the system is saturated, and you are running low on physical RAM
- a single, sequential pass of the data is specified.

Direct I/O frees system resources (including memory). The advantage of direct I/O processing is that memory is not thrashed; the disadvantage is that read-aheads and write-behinds are no longer available to you.

Another advantage to avoiding file cache on systems that are I/O bound is working directly from disk, which cuts in half the amount of I/O from the ETL flow. The advantage of this process is less I/O; a disadvantage is that the file is not readily available in memory. Therefore, if your ETL flow is using the same file repeatedly, and the file is small enough to fit in memory, then accessing the file from disks as opposed to memory that is close, results in the job taking more time to run.

Note: Currently, you can only execute direct I/O on Windows 32-bit systems using SAS®9 with the `SGIO` option and on most UNIX systems by using a file system MOUNT command. For more information, see "SGIO System Option: Windows" in the *SAS OnlineDoc 9.1.3* at support.sas.com/onlinedoc/913/docMainpage.jsp and "Setting Up and Tuning Direct I/O for SAS®9 on AIX" (Bartucca 2005) available at www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100890.

Monitoring SAS Data Integration Studio Flows

Periodically, you should monitor the SAS ETL flows to ensure that they are performing to specification (most importantly, the ability to run in a pre-determined time frame). Experience has shown that, over time, the data volume increases: this might be input data coming into the ETL flow or it might be the back-end data warehouse becoming larger as the historical data is loaded. To ensure that the ETL flow does not fail, you should monitor the flow to verify that it is completing within the desired time frame. Do this by reviewing the SAS log files from the ETL flow to see if they are completing successfully (and with plenty of time to spare), and that there are no large gaps between the real time and the CPU time.

In addition to reviewing the SAS logs, use various operating environment tools or third-party hardware monitoring tools to monitor the computer for any system-resource bottlenecks (such as I/O, memory, or CPU constraints), and to ensure that you have enough disk space to complete the ETL flow. It is always better to monitor the status of your long-running ETL flows, periodically, instead of having to troubleshoot to find out why an ETL flow failed today (ran too long or ran out of disk space), although it has completed, successfully, on other days.

Advanced Topics

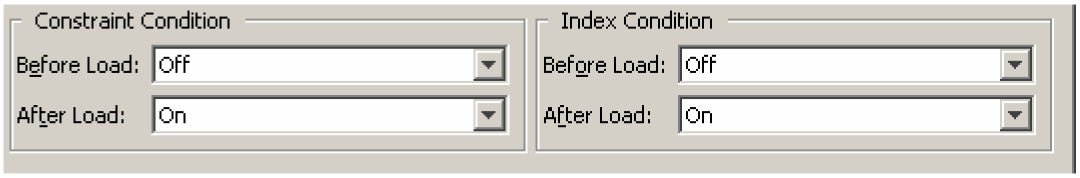
Now that you've tried the basics and the simple tuning techniques, if you are still having trouble with performance or you want to investigate other methods of improving performance, you are ready for the advanced topics. This section discusses topics such as how to modify SAS Data Integration Studio-generated code, how the order of a join can affect performance, how SAS functions are passed to the databases, the best method for getting counts from SAS data sets and the databases, and how to optimally use dates in SAS SQL statements.

Removing Non-Essential Indexes and Constraints During a Load

In some situations, removing non-essential indexes *before* a load and re-creating those indexes *after* the load improves performance. As a general rule, consider removing and re-creating indexes if more than 10% of the data in the table will be re-loaded.

You might also want to temporarily remove key constraints in order to improve performance. If there are a significant number of transactions and the data that is being loaded conforms to the constraints, then removing the constraints from the target *before* a load removes the overhead that is associated with maintaining those constraints during the load.

To control the timing of index and constraint removal, use the options that are available on the **`Load Technique`** tab (Figure 9) of the Table Loader transformation (version 2). There are four settings that enable you to specify the desired conditions for the indexes and constraints before and after the load.



Constraint Condition — Before Load: Off; After Load: On

Index Condition — Before Load: Off; After Load: On

Figure 9. Constraint and Condition Options on Load Technique Tab of Table Loader Transformation

The options that are available will depend on the load technique that you choose. The choices translate to three different tasks: *put on, take off,* and *leave as is*. (For more information, see the product Help.) Select **Off** for the Before Load option(s) and the generated code checks for and removes any indexes (or constraints) that are found, and then loads the table. If an index is required for an update, that index is not removed or it will be added, as needed. Select **On** for the After Load option(s) to have indexes added after the load.

After the table is loaded, if you decide not to build indexes, your physical table will be out of sync with the metadata for the table. Here is a scenario where you might want to leave indexes off during and after loading for performance reasons: Indexes are defined on the table only to improve performance of a query and reporting application that runs after the nightly load. The table is updated multiple times in a series of load steps that appear in separate jobs. None of the load steps need the indexes, and leaving the indexes on impedes the performance of the load. In this scenario, the indexes can be taken off before the first update and left off until after the final update.

Optimizing Join Order and Performance

Sometimes, changing the oder of a join in a multi-stage join can substantially improve the overall elapse time (performance) and resource consumption. The SAS Data Integration Studio SQL Join transformation gives you complete control over the order in which your joins are performed. The default syntax that is generated is implicit inner joins, but you can control this in the transformation.

You might want to use the order of a join to reduce the number of columns in a table that has many columns and rows. The difference in the join processing can reduce the amount of data that must participate in the join. You can use the interface for the SQL Join transformation to change the order of the join.

You can also turn on the DEBUG option in PROC SQL, which enables you to examine the SQL tree in order to determine how processing will be performed.

GROUPINTERNAL Option in PROC MEANS

The `GROUPINTERNAL` option in PROC MEANS tells the procedure not to apply formats to class variables when PROC MEANS groups the values to create combinations of class variables. Using this option can be useful when there are numeric values that do not have a format specified. PROC MEANS converts the numeric values to a character string by using the BEST12 format, then converts this value to a character string, which saves computer resources when the numeric values contain discrete values.

Lookups in a Star Schema

Lookup is frequently used to validate data values against a reference table, to add descriptive text from a code table, or to add values from dimension tables to a fact table in a star schema.

A **star schema** is a method of organizing information in a data warehouse that enables efficient retrieval of business information. A Star Schema consists of a collection of tables that are logically related to each other. Using **foreign key** references, data is organized around a large central table that is called the "fact table", which is related to a set of smaller tables that are called "dimension tables". To load data into the fact table, each record must be loaded with the associated dimension-table, foreign-key entry.

There are several techniques commonly used to perform lookups. Lookups can be made directly against *data on disk* by using join or merge, or against *data in memory* by using a DATA step with SAS Formats or Hash Objects (new in SAS®9). The technique that you choose can have a significant impact on how well your processes perform.

For general lookups, given adequate available memory, the SAS DATA step hash technique is the best all-around choice. With the hash object, a lookup can be made with a composite key, and more than one column of data can be retrieved from the lookup data (tasks not easily accomplished with Formats). The hash object uses less memory than the Format. If the source data for the lookup is large and memory constraints are an issue, the hash object and SAS format techniques might be slow due to system paging, or they might fail completely. Join or Merge lookup techniques might be required.

SAS Data Integration Studio provides several transformations in the transformation library that can be used to perform lookups. Choose the Lookup transformation to perform a hash-object lookup. Choose the Fact Table Lookup transformation to perform a SAS Format lookup. Choose the SQL Join transformation for joining tables.

Hash-Object Lookup

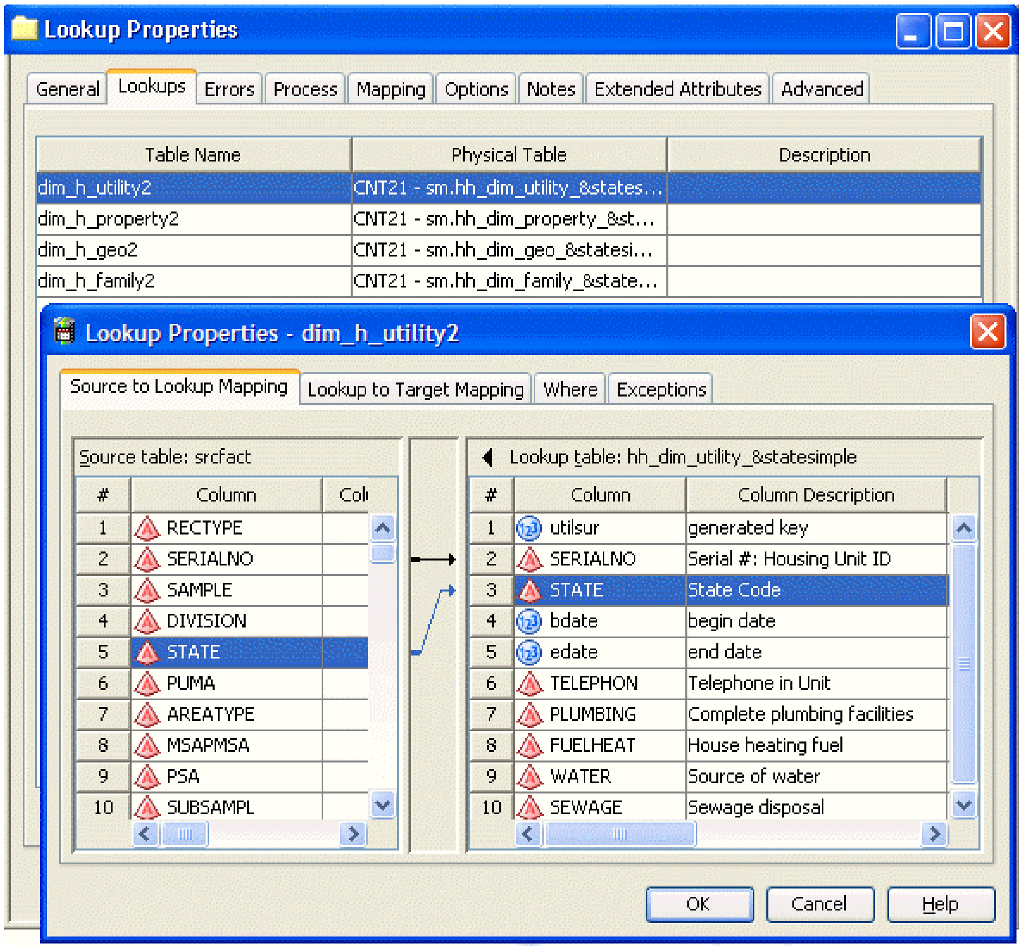
The SAS DATA step hash lookup enables you to store data in an expandable, in-memory table that has fast lookup capabilities. It is similar to a DATA step array, where an index is used to store and retrieve data in the array. While an array uses numeric indexes, a hash table can use character, numeric, or a combination of character and numeric indexes. Unlike an array, the number of values stored in a hash table is not specified when creating the table. A hash table can grow to hold as many values as will fit into memory. When adding data to a hash table, an index, often called a key, is used to find a position in the table to copy the data. When retrieving data from a hash table, a key is given and a fast, nonlinear search occurs to determine if data with the key has been placed in the table. If so, the data that is stored with that key is returned. The hash table is destroyed when the DATA step ends. Hash objects are also scalable because one or more lookups can be performed on a single pass of the data. The hash technique also supports lookup on one or more source-key values and retrieval of one or more target values.

Hash-object lookup is ideal for a one-time lookup step. A hash-object lookup builds rapidly, consumes no disk space, is fast, and it scales well to multiple values. A hash table is a good choice for lookups in unordered data that can fit into memory.

The Lookup transformation in SAS Data Integration Studio leverages the hash technique to perform lookups. The transformation has the following advantages:

- It uses the fast-performing, DATA step hash technique as the lookup algorithm for matching source values to target values.
- It allows multi-column key lookups and multi-column propagation into the target table.
- It supports WHERE-clause extracts on dimension tables for subsetting the number of data records.
- It enables customized error handling.

Figure 10 shows how to configure the Lookup transformation to build fact table data from associated dimension tables. The first step is to configure each lookup table in the Lookup transformation. The **`Source to Lookup Mapping`** tab is used to indicate the columns to use in order to match the incoming source record to the lookup table. Note that multiple columns can be used to perform the match.


Lookup Properties

General | Lookups | Errors | Process | Mapping | Options | Notes | Extended Attributes | Advanced

Table Name	Physical Table	Description
dim_h_utility2	CNT21 - sm.hh_dim_utility_&states...	
dim_h_property2	CNT21 - sm.hh_dim_property_&st...	
dim_h_geo2	CNT21 - sm.hh_dim_geo_&statesi...	
dim_h_family2	CNT21 - sm.hh_dim_family_&state...	

Lookup Properties - dim_h_utility2

Source to Lookup Mapping | Lookup to Target Mapping | Where | Exceptions

Source table: srcfact

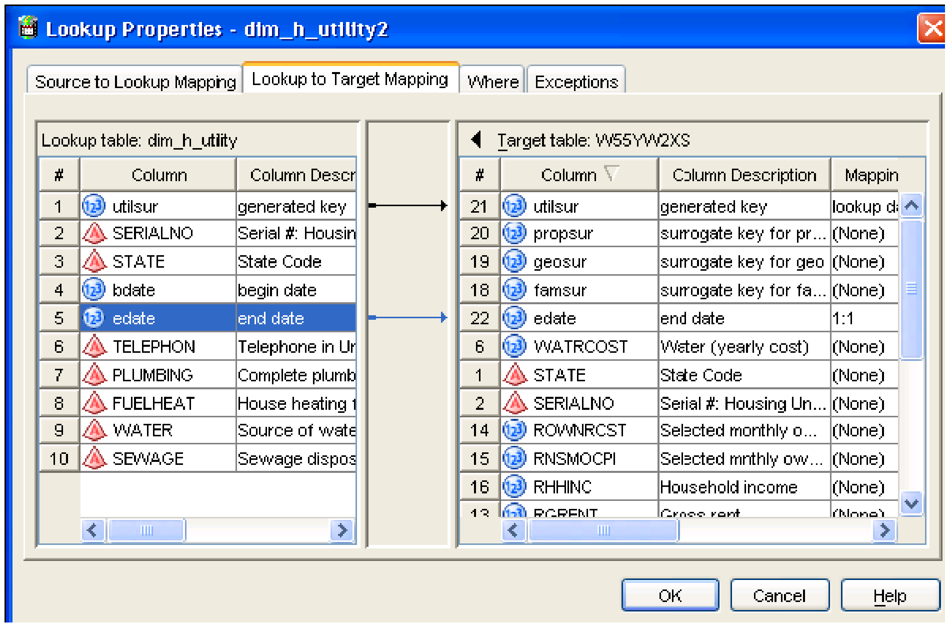
#	Column	Col
1	RECTYPE	
2	SERIALNO	
3	SAMPLE	
4	DIVISION	
5	STATE	
6	PUMA	
7	AREATYPE	
8	MSAPMSA	
9	PSA	
10	SUBSAMPL	

Lookup table: hh_dim_utility_&statesimple

#	Column	Column Description
1	utilsur	generated key
2	SERIALNO	Serial #: Housing Unit ID
3	STATE	State Code
4	bdate	begin date
5	edate	end date
6	TELEPHON	Telephone in Unit
7	PLUMBING	Complete plumbing facilities
8	FUELHEAT	House heating fuel
9	WATER	Source of water
10	SEWAGE	Sewage disposal

Figure 10. Configuring the Lookup Transformation to Build Fact Table Data from Associated Dimension Tables

Figure 11 shows the columns to pull out of the lookup table and propagate to the target table. Because the Lookup transformation supports multi-column keys, any number of attributes can be pulled out of each lookup table and retained in the target table.



Lookup table: dim_h_utility

#	Column	Column Descr
1	utilsur	generated key
2	SERIALNO	Serial #: Housin
3	STATE	State Code
4	bdate	begin date
5	edate	end date
6	TELEPHON	Telephone in Ur
7	PLUMBING	Complete plumb
8	FUELHEAT	House heating f
9	WATER	Source of wate
10	SEWAGE	Sewage dispos

Target table: W55YW2XS

#	Column	Column Description	Mappin
21	utilsur	generated key	lookup d
20	propsur	surrogate key for pr...	(None)
19	geosur	surrogate key for geo	(None)
18	famsur	surrogate key for fa...	(None)
22	edate	end date	1:1
6	WATRCOST	Water (yearly cost)	(None)
1	STATE	State Code	(None)
2	SERIALNO	Serial #: Housing Un...	(None)
14	ROWNRCST	Selected monthly o...	(None)
15	RNSMOCPI	Selected mnthly ow...	(None)
16	RHHINC	Household income	(None)
13	RGRENT	Gross rent	(None)

Figure 11. Selecting Lookup Table Columns to Propagate the Target Table

The Lookup transformation is especially useful when the complexity of the process increases because the hash-based lookup easily handles multi-column keys and multiple lookups without degrading performance. For example, suppose that each fact in a fact table requires a composite foreign key from the dimension table. Adding the complexity to include these associated fields can degrade the performance of the process when using methods such as joins or formats. It can also increase the complexity of the code to handle the process. Using hash lookup, an additional table is added as another lookup entry and matching can occur on one or more columns.

The following example shows the DATA step code for a hash-based lookup. This example uses a multi-column lookup key and pulls two columns into the target table.

```
if _n_=1 then
do;
           /* Build hash h3 from lookup table CMAIN.Household_income */
           declare hash h3(dataset:"CMAIN.Household_income");

           /* lookup on a 2 column key */
           h3.defineKey("SERIALNO", "STATE");

           /* pull two columns out into the target data set */
           h3.defineData("INCOME", "YEARS");

           /* lookup complete */
           h3.defineDone();
end;
```

Sample Test Results

To test performance, an example lookup was coded to perform the search in four ways: using a format, SET with KEY=, MERGE with BY, and HASH lookup. The results are shown in the table below.

Real Time to Perform Lookup

SET with KEY= 108.14s
Build Index 11.33s
Search 96.81s
Format and PUT() 89.03s
Build Format 67.32s
Search 21.71s
MERGE with BY 35.78s
Sort Data Sets 31.60s
Search 4.18s
Hash Table 18.35s
Load and Search 18.35s

For more information, see "The DATA Step in SAS 9: What's New?" (Seckosky 2004) at support.sas.com/rnd/base/topics/datastep/dsv9-sugi-v3.pdf.

Format-Based Lookup

The FORMAT procedure provides another method for performing lookups. PROC FORMAT creates a permanent lookup table that is saved in the SAS format library for use later in the job or in subsequent jobs. A step that references the format reads it into memory; the source table is not accessed. However, compared to the hash object, formats require additional processing and disk space to create and store the format. Sorting and I/O overhead to create the format can be costly. Thus for a one-time lookup, the format method is less desirable to use than hash.

The following example illustrates how to create a FORMAT-based lookup. You can use user-defined formats to display or write-out coded values in raw data. This technique enables you to re-write If-Then/Else trees and replace them with a single line of code. For example, assume that you have a set of discount factors that are stored in the user-defined format $DISC.

```
proc format;

value $disc

‘ABC’ = 0.20

‘DEF’ = 0.25

‘XYZ’ = 0.00

other = 0.00;
```

You can replace code that looks like this:

```
if vendor = ‘ABC’ then discount = 0.20;

else if vendor = ‘DEF’ then discount = 0.25;

else if vendor = ‘XYZ’ then discount = 0.00;
```

with a single statement that looks like this:

```
discount = input(put( vendor, $disc. ),best32.);
```

This technique has the added advantage of separating the data (that is, the table of discount factors) from the code. If you need to add or to change the discount values for your vendors, you change that data outside of the DATA step and leave your existing DATA-step code alone. Note that the PUT( ) function performs the lookup and returns a character string. The INPUT( ) function (as used here) converts that character value to a numeric value.

You can also create a user-defined format from an existing data set or database table. PROC FORMAT provides an analog to the `CNTLOUT=` option called `CNTLIN=`, which loads a user-defined format from a data set. For example, you have an existing data set, DISCOUNT, that has two columns: VENDOR and DISCOUNT. You can build a suitable `CNTLIN=` data set from the DISCOUNT data set as follows:

```
data cntlin(

keep = fmtname type hlo start label );

retain fmtname ‘disc’ type ‘C’;

set discount end = lastrec;

start = vendor; label = put( discount, 6.2 );

output;

if lastrec then do;

hlo = ‘O’; label = ‘0.00’;

output;

end;

run;
```

For more information about formats, see "*Eight PROC FORMAT Gems*" (Shoemaker 2001) available at www.suk.sas.com/proceedings/26/26/p062-26.pdf.

Performance Considerations with Lookup

1. When using the DATA step hash technique, verify that you have enough available RAM to hold the hash table. Adjust the value for the `MEMSIZE=` option to be large enough to hold the in-memory hash table and support other SAS memory needs. However, be aware of increasing the value of the `MEMSIZE=` option beyond the amount of available physical memory because system paging will occur and degrade job performance.

2. In order to minimize the amount of memory that is required to hold the lookup objects, create the hash object or format on the smaller table.

3. If you are performing processes in parallel, ensure that each parallel job has sufficient physical RAM to hold the format or hash object, because these structures are NOT shared between SAS jobs.

4. A format exists for the life of the SAS session; a hash object exists only within the single DATA step in which it is created. If lookups against a table key is a one-time event in your ETL flow, use the hash object. If several steps perform identical lookups, consider using the format-based lookup. One read of the lookup's source table to create a format in memory is cheaper than the multiple reads that would be required to construct hash objects, repeatedly, in the separate steps. The difference in performance can be discernible when the lookup's source table is large.

5. The hash object consumes less memory than a format. If you are using a format and experiencing system paging, try replacing the format with a functionally equivalent hash object.

6. The hash object easily supports composite keys and composite data fields. You must "coerce" a format to support more than a single, key column and single, data column.

7. For large tables that reside in a database, you can use database-specific SQL to perform the lookup in the database. Using the hash object or a format causes the database tables to be imported into SAS for the lookup. Network and I/O overhead make this method slow relative to performing the work in the database.

Processing Level Performed by a Relational Database

When working with relational database source data, it can be performance-critical to push work to the database. For large data extracts, examine the generated SQL to verify that filtering and other processing is passed to the database.

Use the SASTRACE= option to display the SQL that is passed to the database by a SAS/ACCESS engine. The SASTRACE option displays SQL, some API call information, and database return codes or messages in your SAS log file. For more information, see "SASTRACE= System Option" in *SAS/ACCESS® 9.1.3 for Relational Databases: Reference* available at support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/access_rdbref_9297.pdf.

The list of SAS functions that are translated to the database equivalent is specific to each SAS/ACCESS product.

Enable tracing by using the following OPTIONS statement:

```
option SASTRACE=”,,,d” no$stsuffix SASTRACELOC=saslog;
```

Disable tracing by using the following OPTIONS statement:

```
option SASTRACE=”,,,”;
```

SAS/ACCESS engines send SQL for both *prepare* and *execute* on the database. *Prepare-only SQL* is inexpensive on the database. An example of a *Prepare-only SQL* is the "SELECT * from tablename" prepare to determine if a table exists and to retrieve column metadata.

Executed SQL contains any WHERE filters, joins, or other complex queries that are passed to the database. *Executed SQL* triggers substantive database activity by transferring rows from the database to SAS.

Sometimes, warnings and errors that are generated in trace output are not a cause for concern. The warnings and errors might indicate that the SQL was not handled by the database, and it was passed back to SAS to process.

UPCASE and LOWCASE Functions versus UPPER and LOWER Functions

SAS/ACCESS engines will attempt to pass SAS functions to the underlying relational database if the database provides an equivalent function. If you need to use a SAS function in a WHERE clause, it is important to consider if the SAS function can be passed to the database by the SAS/ACCESS interface that is being used.

The SAS functions UPCASE and LOWCASE are similar in functionality to the SQL UPPER and LOWER string functions. SAS SQL treats UPPER and LOWER differently than it treats UPCASE and LOWCASE in the SQL that is generated to a database. The UPCASE and LOWCASE functions are preferred because they can be translated to database-specific equivalent functions. WHERE-clause filtering that is performed on the database for UPCASE and LOWCASE causes a significant difference in processing time.

The first example code uses the UPPER and LOWER string functions. This is illustrated by the use of the `SASTRACE` option with SAS/ACCESS to DB2.

```
options sastrace=",,,d" sastraceloc=saslog no$stsuffix;


proc sql;

select Component_Id from dbms.project

where UPPER(Complete_code)  = 'PENDING' or LOWER(lead) = 'valance';

quit;
```

```
DB2_4: Prepared:

SELECT * FROM MYDB2.PROJECT FOR READ ONLY


DB2: COMMIT performed on connection 0.


DB2_5: Executed:

Prepared statement DB2_4


NOTE: PROCEDURE SQL used (Total process time):

      real time           3.35 seconds

      user cpu time       3.01 seconds
```

Note: In the preceding example, notice that the WHERE-clause processing was not passed to the database, therefore, all the data was processed in SAS.

The next example shows the same code that is used in the first example, but this code specifies the UPCASE and LOWCASE functions instead of the UPPER and LOWER string functions.

```
proc sql;

select Component_Id from dbms.project

where UPCASE(Complete_code)  = 'PENDING' or LOWCASE(lead) = 'valance';

quit;


DB2: AUTOCOMMIT turned ON for connection id 0


DB2_1: Prepared:

SELECT * FROM MYDB2.PROJECT FOR READ ONLY


DB2: COMMIT performed on connection 0.


DB2_2: Prepared:

SELECT  "COMPONENT_ID", "COMPLETE_CODE", "LEAD"  FROM MYDB2.PROJECT WHERE
```

```
( ({fn UCASE( "COMPLETE_CODE")} = 'PENDING' ) OR  ({fn LCASE( "LEAD")} =
'valance' )  ) FOR READ ONLY



DB2_3: Executed:
Prepared statement DB2_2

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.20 seconds
      user cpu time       0.04 seconds
```

Note: In the preceding example, notice that the WHERE-clause processing was passed to the database.

You can also turn on the DEBUG option in SAS Data Integration Studio version 2 of the SAS SQL Join transformation; this will generate the SASTRACE option.

Getting Counts of the Number of Rows in a Table

In an ETL flow, you might need a count of the number of rows in a table. There are several ways to get a count; some ways are more efficient than others, depending on the data source.

The following examples show ways to count rows in tables for both SAS data sets and relational database tables.

Row Counts in SAS Data Sets

There are easy and efficient ways to get counts for SAS data sets. You can use either the CONTENTS procedure or a dictionary call in PROC SQL to get a count. Following is an example that uses PROC CONTENTS.

```
proc contents noprint
    data = mydata.test
    out = mydata.contents;
run;
data _null_;
    set mydata.contents;
    if _n_ = 1 then do;
```

```
            call symput("counter", left(trim(put(nobs,best.))));

            stop;

        end;

    run;
```

If you try to get a count from a relational database system, PROC CONTENTS returns a value of -1.

Using the SET statement with the `NOBS=` option, as shown in the following example, might be faster because PROC CONTENTS is eliminated.

```
    data _null_;

      if 0 then set mydata.test nobs = nobs;

      call symputx("counter", nobs);

      stop;

    run;
```

You can also use a dictionary call in PROC SQL. If you use this method, subtract the deleted observations (DELOBS) from the total number of observations (NOBS). Following is an example:

```
    proc sql noprint;

        select nobs - delobs

        into :counter

        from dictionary.tables

        where

            libname = "MYLIB" and

            memname = "TEST" and

            memtype = "DATA"

        ;

    quit;
```

If the dictionary call returns a missing value, then it was unsuccessful in getting a count from the data source. See the section "Row Counts in Databases".

Note: In SAS 9.1.3 Service Pack 4 or later, the **COUNT(*)** statement in SAS SQL, which is recommended for databases, can now be used efficiently with SAS data files. SAS will retrieve the number of observations from the header.

Row Counts in Databases

Relational databases do not have the number of rows in a table readily available. As a result, when asked for a count, SAS/ACCESS engines return a value of -1 when PROC CONTENTS is used or a missing value when PROC SQL is used. To get a row count in a relational database, use the SELECT COUNT(*) command. This command might result in a full-table scan, but it is the recommended method of getting counts, and, usually, relational databases handle full-table counts with an acceptable amount of efficiency. Here is an example that uses PROC SQL:

```
proc sql noprint;

    select count(*)

    into :counter

    from mydb.test;

quit;
```

Bulk Loading the Relational Database

One of the fastest ways to insert large data volumes into a relational database, when you use the SAS/ACCESS engine, is to use the bulk-loading capabilities of the database. By default, the SAS/ACCESS engines load data into tables by preparing an SQL INSERT statement, executing the INSERT statement for each row, and periodically issuing a COMMIT. If you specify `BULKLOAD=YES` as a data set or a LIBNAME option, a database bulk-load method is used. This can significantly enhance performance, especially, when database tables are indexed. Additional information is available in the SAS/ACCESS online documentation available at support.sas.com/documentation/onlinedoc/91pdf/index_913.html#access and in database vendor bulk-load documentation. (The term "bulk load" is not useful in finding or navigating database documentation; instead, look for the load methods, such as SQL*Loader and FastLoad.)

Note: Some SAS/ACCESS engines implement `BULKLOAD=YES` as a LIBNAME option, however, this is not supported across all SAS/ACCESS engines. Therefore, specify `BULKLOAD=` as a data set option for reliable behavior across SAS/ACCESS engines.

The Use Bulkload for Uploading and Bulkload Options properties are available on the Property Sheets in SAS Data Integration Studio for each table in a version 2 SQL Join transformation. The Use Bulkload for Uploading option to bulk-load tables only applies to source tables in a heterogeneous join. Also, the user must upload the table to the DBMS where the join is performed. Use Bulkload for Uploading is only a valid option when the source table is being uploaded to the DBMS to create a homogeneous join. The Bulkload to DBMS property applies to target tables and turns bulk loading on and off. The Bulkload to DBMS property is not valid when the target table has the *Pass Through* property on the SQL Properties pane set to **Yes**.

SAS Date Functions

If your ETL process needs to convert or assign date and datetime values, your first instinct might be to use a SAS date function. These functions are very powerful, but there are some issues you need to be aware of when you use a SAS date function (or any function that exists only in SAS):

- Function calls incur overhead for every row that is processed via the ETL step. This overhead should be avoided for WHERE filtering or other iterative processes when the function's desired returned value is, in effect, a constant value across all rows. The cost is negligible for small tables, **but it is substantial if there are millions of rows**.
- During a long-running process, SAS date functions can yield unintended or undesirable results. For example, the value that is returned by the TODAY( ) function will change if the job runs past midnight into the following day.
- SAS/ACCESS engines might not support the passing down of these functions for evaluation on the relational database. The result is that each record will be read into SAS so that the function can be processed.
- SAS/ACCESS engines might not support passing some WHERE clause expressions that contain date and time variables.

One alternative to using a SAS function directly in a filter or iterative process is to pre-evaluate the function expression in a SAS macro variable definition. Then, the SAS macro variable can be used in a direct comparison in a SAS WHERE clause or in assignments in a SAS DATA step.

The following examples show the performance gain when comparing the two techniques. The following two DATA steps assign a date value to a variable. The first DATA step accomplishes the assignment by using the SAS TODAY function. The second DATA step uses a macro variable that has been assigned the value of the same SAS TODAY function that is used in the first DATA step. The assignment is repeated one million times in each DATA step to show that the processing overhead of the function that is used in the DATA step increases with each iteration.

```

/* today() function used directly in variable assignment */
data _null_;
   format date_var date9.;
   do n=1 to 1000000;
      date_var = today();
   end;
   put date_var=;
run;

date_var=13SEP2005
NOTE: DATA statement used (Total process time):

      real time           1.04 seconds
      cpu time            0.98 seconds
```

The next example replaces DATEVAR=TODAY() with a macro variable that contains today's date. This assignment is made only once.

```

      /* macro variable for subsequent variable assignment */
      %let target_date=%sysfunc(today());

      data _null_;
         format date_var date9.;
         do n=1 to 1000000;
            date_var = &target_date;
         end;
         put date_var=;
      run;

date_var=13SEP2005
NOTE: DATA statement used (Total process time):

      real time           0.03 seconds
      cpu time            0.03 seconds
```

The preceding technique should work effectively with any SAS data source.

Views or Physical Tables for ETL Flows

In general, each step in an ETL flow creates an output table that becomes the input of the next step in the flow. Decide which of the following would be best for transferring data between steps in the flow:

- write the output for a step to disk (in the form of SAS data files or relational database tables)
- create views that process input and pass the output directly to the next step (with the intent of bypassing some writes to disk).

Note that SAS supports two types of views: SQL views and DATA step views. These two types of views can behave differently. Switching from views to physical tables or tables to views, sometimes makes little difference in an ETL flow. At other times, improvements can be significant. Here are some useful tips.

- If the data that is defined by a view is only referenced once in an ETL flow, then a view is usually appropriate.
- If the data that is defined by a view is referenced multiple times in an ETL flow, then putting the data into a physical table will probably improve overall performance. When accessing a view, SAS must execute the underlying code repeatedly, that is, each time the view is accessed.
- If the view is referenced once in an ETL flow but the procedure performs multiple passes of the input, then consider using a physical table.
- If the view is SQL and referenced once but that view references another SQL view, then consider using a physical table. SAS SQL optimization can be less effective when views are nested. This is especially true if the view performs joins or accesses relational database sources.
- If the view is SQL and involves a multi-way join, the view is subject to the performance limitations and disk space considerations that are discussed in the section "SAS SQL Joins".
- When creating physical tables, SAS optimizes where the columns are placed on disk and in memory in order to avoid unnecessary padding. However, with views, the columns are placed in memory in the order in which they are specified in the SELECT statement. You need to ensure that there is no extra padding added that would force the non-character column to the next 8-byte aligned boundary. This extra padding causes the record length to become very wide, which results in extra I/O when you manipulate the record.
- SAS physical tables honor short numerics (less than 8 bytes in length), however, views expand all short numerics to 8 bytes. Therefore, three short numerics that are 4 bytes in length (for a total of 12 bytes in a SAS data set) become 24 bytes when accessed by using a view. .

Assess the overall impact to your ETL flow if you make changes based on these tips. In some circumstances, you might find that you have to sacrifice performance in order to conserve disk space or vice versa.

Views or Physical Tables for SAS Data Integration Studio Flows

The process flows in SAS Data Integration Studio jobs include icons for sources, targets, and transformations. In order to facilitate performance in process flows, the output of many transformations can be either a physical table or a view. Each format has its advantages, as noted in the preceding section.

WORK tables that are displayed in the Process Editor have an option called "**Create View**"on the menu that results when you right-click. Not all transformations can support a view, but the following are some of the transformations that enable you to specify a view format or a physical table format for their output tables.

- Append
- Extract
- Data Validation
- SQL Join
- Library Contents
- Lookup

Loading Fact and Dimension Tables When Working with Dimensional Models

Fact and dimension tables can be used extensively in data integration. You need an understanding of what they are and how to use them.

Overview of Dimensional Modeling

Dimensional modeling is a methodology for logically modeling the data for easy query access. Dimensional models contain a single fact table that is surrounded by dimension tables. Dimension tables record a category of information such as time, customer information, and so on. Usually, the information that is stored in dimension tables does not change frequently. Fact tables contain information that is related to the measures of interest for example, sales amount, which is measured at a time interval. This measure is stored in the fact table with the appropriate granularity such as hour, day, or week.

Usually, the data that is coming into the dimensional model is collected from many sources. This data might not have unique identifiers across all its sources. For example, the data might have several different customer identifiers (also known as **business keys**) that are the same for different customers' last names. Each of these entries indicates a different customer, therefore, each entry should have a unique value assigned to it in order to differentiate among customers. The process of creating a complete set of unique identifiers when storing information in a dimensional model is called **generating a surrogate key**. Usually, the **primary key,** which is stored in a dimension table, is a unique surrogate key that is assigned to each entry in the dimension table. The business key is stored in the dimension table along with the unique surrogate key, so that each entry is unique in the dimension table.

The fact table contains foreign keys that are imported from each of the dimensions that are related to it. The primary keys are stored in the related dimension tables. Key relationships among tables are **registered** in the **metadata repository** when the metadata about tables is captured in SAS Data Integration Studio via the source designers or a metadata import. Key relationships can also be defined and viewed on the `Keys` tab of a table's property window. When information about dimension tables is captured via the source designers or a metadata import, be sure to register all related tables together so that the primary and the foreign keys are registered.

Transformations for Generating Keys and Loading Dimension Tables

SAS Data Integration Studio provides several transformations that generate surrogate keys and load dimension tables. It is recommended that you use the Slowly Changing Dimensions (SCD) transformation when you are loading dimension tables. This transformation is located in the Data Transforms Process Library named SCD Type2 Loader. The SCD Type2 Loader transformation contains algorithms that enable creation of a unique **generated key** based on an incoming business key and tracks changes to the incoming data. If the incoming data contains changes to the values in an observation, you might want to retain this history in the dimension table so that you can track these changes over time. The SCD Type2 Loader transformation provides several ways of retaining this type of historic information when loading a dimension table.

The Surrogate Key Generator transformation is also available when only a **unique key** value has to be generated for a given set of data. This transformation is also available in the Data Transforms Process Library folder.

Transformations for Loading Fact Tables and Matching Fact Tables to Dimension Tables

For loading fact tables and for performing key matching with dimension tables, SAS Data Integration Studio provides the following transformations: Lookup and Fact Table Lookup. The preferred transformation to use when loading fact tables is the Lookup transformation, which is located in the Data Transformations folder of the Process Library. The Lookup transformation has several advantages:

- It uses the fast-performing DATA-step hashing technique as the lookup algorithm for matching source values from the fact table to the dimension tables.
- It enables multi-column key lookups.
- It enables users to customize error-handling when problems occur in the source or dimension records such as missing or invalid values.

Fact and Dimension Tables: Order of Loading Is Important

Because of the relationships that exist between fact and dimension tables, the order in which you load data into these tables is important. Because the primary key in the relationship is contained in the dimension table, dimension tables must be loaded first when you are loading data into a dimensional model set of tables. After the data has been loaded into the dimensional tables, facts or measures can be loaded into the fact table. In the incoming data for facts, the business key in each source row is usually matched with values in the associated dimension tables. The matching row in the dimension table provides a generated key value that is added to the specified target column for that new target row in the fact table. This directly associates each fact with the appropriate dimension tables in the dimensional model and enables fast retrieval of information in response to queries. Usually, to retrieve the data, only one level of join (fact table to dimension table) has to occur.

SAS Scalable Performance Data Server Star Joins

You can use the SAS Data Integration Studio SQL Join transformation to construct SAS Scalable Performance Data Server (SAS SPD Server) star joins when you use SAS SPD Server 4.2 or later. Star joins are useful when you query information from dimensional models that contain two or more dimension tables which surround a centralized fact table that is referred to as a **star schema**. SAS SPD Server star joins are queries that validate, optimize, and execute SQL queries in the SAS SPD Server database for upgraded performance. If the star join is not used, the SQL is processed in SAS SPD Server using pair-wise joins that require one step for each table in order to complete the join. When the SAS SPD Server options are set, the star join is enabled.

To enable a star join in the SAS SPD Server, the following requirements must be met:

- All dimension tables must surround a single fact table.
- Dimension-to-fact table joins must be equal joins, and there should be one join per dimension table.
- You must have at least two or more dimension tables in the join condition.
- The fact table must have at least one subsetting condition placed on it.
- All subsetting and join conditions must be specified in the WHERE clause.
- Star join optimization must be enabled by setting options in the SAS SPD Server library.

To enable star join optimization, code that will run on the generated Pass SAS SPD Server system library MUST have the following settings:

```
LIBGEN=YES

IP=YES
```

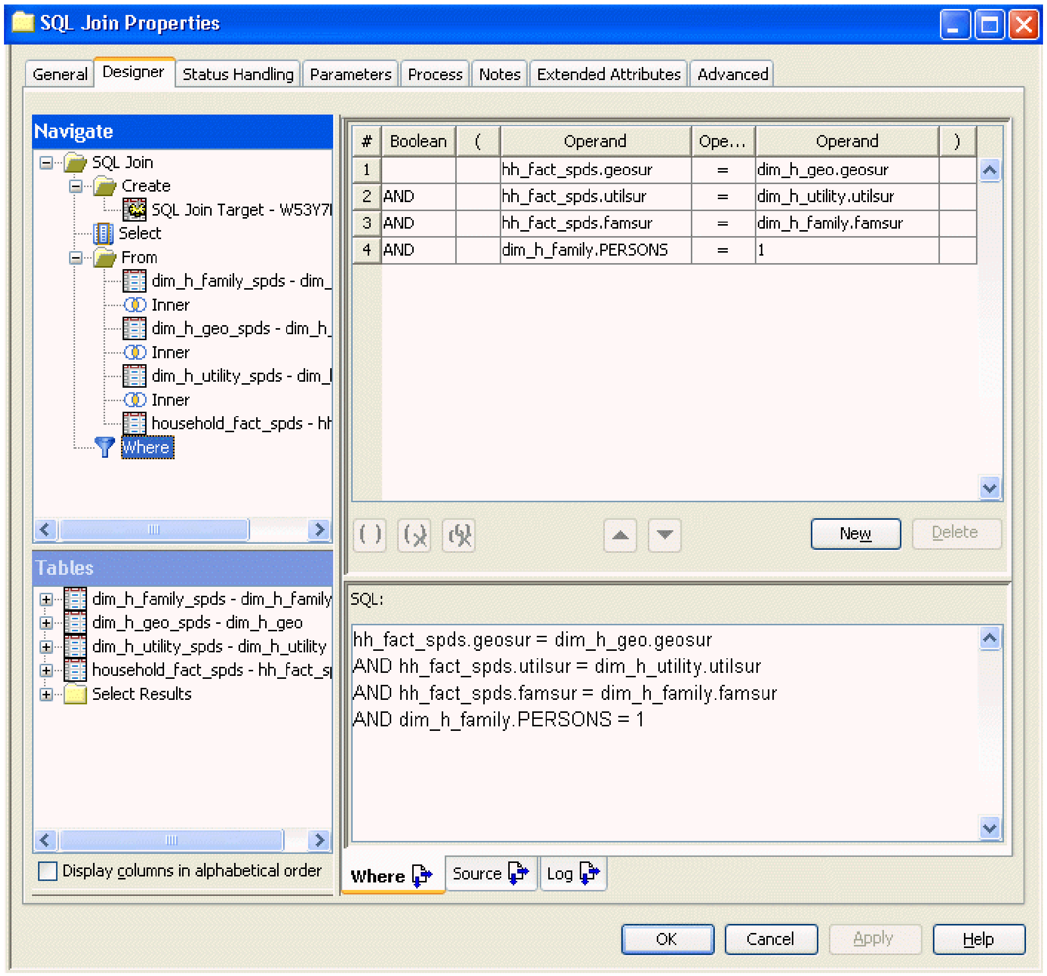
When correctly configured, the following output is generated in the log:

```
SPDS_NOTE: STARJOIN optimization used in SQL execution
```

Here is an example of a WHERE clause that enables a SAS SPD Server star join optimization:

```
/* dimension1 equi-joined on the fact table */

WHERE hh_fact_spds.geosur = dim_h_geo.geosur AND


/* dimension2 equi-joined on the fact table */

     hh_fact_spds.utilsur = dim_h_utility.utilsur AND


/* dimension3 equi-joined on the fact table */

     hh_fact_spds.famsur = dim_h_family.famsur AND



/* subsetting condition on the fact table */

dim_h_family.PERSONS = 1
```

The **`Where`** tab of the SQL Join transformation (see bottom of right panel on the Designer tab in the SQL Join Properties window) can be used to build the condition that is expressed in the preceding WHERE clause. Figure 12 shows how the **`Where`** tab depicts the preceding WHERE clause.



#	Boolean	(	Operand	Ope...	Operand	)
1			hh_fact_spds.geosur	=	dim_h_geo.geosur	
2	AND		hh_fact_spds.utilsur	=	dim_h_utility.utilsur	
3	AND		hh_fact_spds.famsur	=	dim_h_family.famsur	
4	AND		dim_h_family.PERSONS	=	1	

```
hh_fact_spds.geosur = dim_h_geo.geosur
AND hh_fact_spds.utilsur = dim_h_utility.utilsur
AND hh_fact_spds.famsur = dim_h_family.famsur
AND dim_h_family.PERSONS = 1
```

Figure 12. The Where Tab in SQL Join Transformation

Note: The SAS Scalable Performance Data Server requires all subsetting to be implemented on the **Where** tab in the SQL Join transformation. For more information about SAS Scalable Performance Data Server support for star joins, see the *SAS Scalable Performance Data Server 4.4: User's Guide* at support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/scalable_ug_9722.pdf.

Parallel ETL Processing

Sometimes it's possible to divide an ETL flow into multiple parts and execute the parts in parallel in order to reduce overall run time. The success of the parallel execution depends on the ability to divide the process or task into independent parts, the availability of the hardware resources, and the performance benefit analysis of converting to a parallel process.

The secret of effective parallel processing is to know when parallel processing will be effective in improving performance. To run ETL code in parallel requires the designer to divide the task into smaller parts that do not have dependencies. An example of a dependency is where any row in a table cannot be processed until after its preceding row is processed. When a task must be executed in a sequential fashion (that is, one row at a time), that task cannot be divided into individual tasks to leverage parallelism. However, if all the rows in a table *can* be processed independently, without requiring information from another row in the table, then this table can be divided and run in parallel.

It is important to note that your total parallel process run time includes setup, parallel execution, and merging results. Some tasks can be divided in order to run in parallel, but the effort to divide the task or merge the results can take more time than the original sequential execution. Therefore, be sure to estimate the benefit of parallel execution by analyzing the parallel task, set-up time, and the resources that are available before you begin. Your run-time estimate will help you to determine when parallel processing might be beneficial or cost effective.

Software and Hardware Setups for Executing ETL Code in Parallel

Both the software and the hardware must be set up correctly in order to execute ETL code in parallel. The software can be modified manually or tools such as SAS Data Integration Studio can be used to add parallel capability. More importantly, it might be necessary to modify the SAS server setup and the hardware environment in order to gain any benefit from running a task in parallel. Incorrect server setup can quickly mitigate any benefits that might result from running a task in parallel. Setup changes usually involve modifying both the software and the hardware environment.

Software Setup

There are several components built into SAS products to help with parallelism. In SAS Data Integration Studio, looping capability was introduced to help facilitate the conversion of a program to run in parallel. On the SAS server, there are mechanisms to spawn processes across multiple servers and control when they are launched. Best practices for developing ETL flow, load balancing, workspace pooling, and scheduling are given in the following product guides:

- "Enabling Parallel Execution of Process Flows", *SAS® Data Integration Studio 3.3: User's Guide* available at support.sas.com/documentation/onlinedoc/etls/usage33.pdf.
- "Supporting Grid Computing", *SAS® 9.1.3 Intelligence Platform: System Administration Guide* available at support.sas.com/documentation/configuration/bisag.pdf.

Hardware Setup

It is very important to ensure that the hardware has enough resources and is correctly set up when you execute tasks in parallel. The resources that you should look at are: network, I/O, memory, and processors (CPUs).

Network

A network is used when an application uses resources across several computers or nodes. If there is extensive communication among server nodes, it is important to ensure that there will be enough network bandwidth to facilitate communication between processes and grid nodes. You should minimize the amount of data that is transferred on the network for any parallel processing tasks. Parallel processes that must get to a resource on the same system can do this at internal hardware speeds; however, when a resource is spread across the network, it takes significantly more time to get to. If there is a need to transfer large amounts of data to remote resources, consider high-speed networks in which multiple network lines are truncated together for increased throughput. If more throughput for data is needed, consider moving data onto a high-speed storage network (for additional information about this topic, see the "I/O" section that follows).

I/O

I/O performance is a critical factor in parallel performance of SAS applications. If a single threaded process already results in poor I/O performance, performance will be further degraded if you try to run the process in parallel. Before adding parallelism to your ETL flow, it is important that you determine the I/O requirements of the ETL flow. The best practice is to run the program in single threaded mode (one process) and monitor its I/O usage. Learn where the task reads and writes data and how much throughput it uses during execution. System tools are useful for monitoring I/O usage. To estimate the overall I/O requirement for running in parallel, multiply the single process I/O requirement by the number of simultaneous parts that you plan to execute. The I/O requirement can quickly exceed the capability of the system as more simultaneous processes are added to the task, especially if there is a heavy I/O component such as PROC SORT.

Memory

It's easy to forget about memory as an ETL requirement. Memory can quickly become an issue as more simultaneous ETL tasks are added to the system. For example, the Lookup transformation in SAS Data Integration Studio creates a hash table in memory for each dimension table that it references. If the application is looking up values in multiple dimension tables and the tables are of significant size, the application can quickly use up significant amounts of memory. Keep in mind that EVERY simultaneous process will use this same amount of memory by creating its own independent hash tables, because memory cannot be shared between SAS processes. A large dimension table (for example, 10 gigabytes) would require up to 10 gigabytes of random access memory (RAM) per SAS ETL process that executes in parallel. The amount of memory that is actually used will depend on SAS settings such as MEMSIZE, but it is important that you think about memory utilization during coding.

CPU

If the task is a CPU-intensive ETL task (that is, it has many variable calculations or comparisons), then be sure that you know the number of processors that are available in the environment. Make sure that you do not launch so many tasks in parallel that the resources required are larger than the amount of resources available. CPU utilization can be monitored with system performance tools as available for all the operating systems. As with I/O-intensive applications, it's a good idea to estimate the CPU needs by running one sample task in order to determine the CPU requirements.

System Throttles

If the amount of resources cannot be pre-determined, there are SAS Data Integration Studio controls and controls in schedules that can help "throttle" a parallel execution so it won't overrun the available resources. SAS provides a significant number of execution controls in the SAS Data Integration Studio Loop transformation. SAS Server load balancing can also be achieved with the support of third-party scheduling products such as Platform LSF software. By using these "throttles", it is possible to limit the exact number of tasks that are spawned in parallel. Each task can be controlled or launched on different servers based on user profiles or system utilization. Many operating systems also have controls that can help limit the amount of resources that are given to a specific software server or user process. For more information, see the following documentation:

- The "Loop Transformation Guide", "Loop Properties", and "Loop Options Tab" sections in the *SAS® Data Integration Studio 3.3: User's Guide* available at support.sas.com/documentation/onlinedoc/etls/usage33.pdf.
- The topic of scheduling in the *SAS® 9.1.3 Intelligence Platform: System Administration Guide* available at support.sas.com/documentation/configuration/bisag.pdf.

File System Requirements for Grid Computing

In most cases, grid computing is set up to use multiple instances of an operating system—either as independent computers on a network or as domains or LPARs (logical partitions) on a single, larger computer. This is good for processing, but it doesn't take into consideration how the computers will share data. Figure 13 shows an example of how to configure a shared file system with multiple computers.

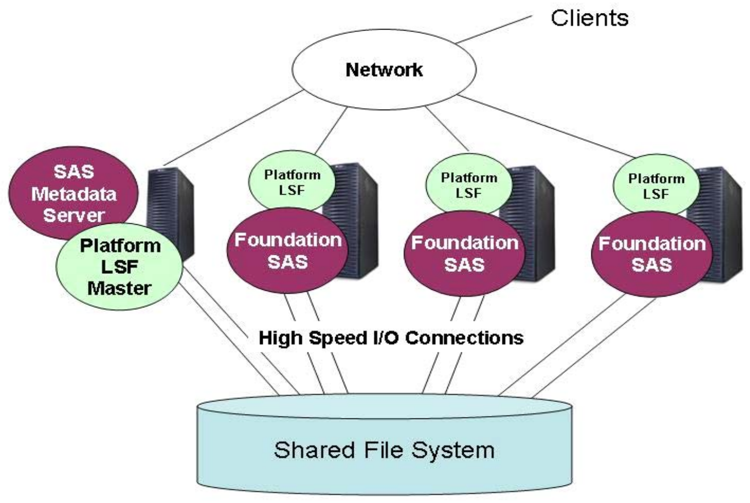


Figure 13. Grid-Computing Environment Configuration Example

If your environment is a collection of independent computers, you need to ensure that all the computers have equal access to file systems with data files that will be shared. It is also important that each parallel process has equivalent access to all data. You can set up shared file systems by using tools that are available on most operating environments, or you can use a clustered file system that is available from the manufacturer of your operating environment or from third parties. Usually, you get more flexibility and I/O throughput from clustered file systems than you get from the shared file systems that come with an operating environment. In either case, it is best to have a thorough understanding of the I/O throughput requirements for your SAS application. It is recommended that you work with your hardware and software vendors to determine the best shared file system implementation for your environment and solution needs.

How to Approach a Parallel ETL Effort

Recently, SAS executed various ETL grid tests across four large UNIX servers that shared one, very large, clustered file system. The file system was connected with equal bandwidth to all grid nodes. During the tests, one of the major tasks was to execute a Star Schema build in parallel to run across the grid. The following process was followed for this task.

1. Analyze the job to learn if it can run in parallel.
2. Determine whether running the job in parallel provides significant improvement in performance.
3. Determine if it's possible to divide the job into parts that can be executed simultaneously.
4. Verify that the hardware environment can effectively execute multiple tasks in parallel.
5. Evaluate I/O, Memory, Network, Bandwidth.
6. Design the work flow.
7. Evaluate whether the estimated run time is worth the effort.
8. Determine if the parallel process will save money.
9. Determine if the parallel process will save time.
10. Determine if the parallel process will fit in your ETL window.
11. Build a small test case on a small part of the data.
12. Run one task in stand-alone mode to get a baseline.
13. Slowly increase the number of tasks and monitor for bottlenecks.
14. Fix issues (hardware, software, code logic, and so on) as they appear.
15. Run the parallel process against the full-sized volume of data; test and validate the results.
16. Tune hardware and software.
17. Run Steps 6 through 10 until performance is acceptable.

For more information, see "Appendix 2: A Sample Scenario for Using Parallel ETL Processing".

Appendix 1: Customizing or Replacing Generated Code in SAS Data Integration Studio

Here are multiple ways you can customize or replace the code in SAS Data Integration Studio.

- Modify the SAS config file, autoexec file or modify the SAS start for the SAS Workspace Server or servers that will execute SAS Data Integration Studio ETL flows.
- Specify options on the **`Code Generation`** tab in the General Options window.
- Add SAS code to the **`Pre and Post Processing`** tab in the Properties window for a job.
- Specify SAS system options for many transformations.
- Write customized code to replace the generated code for a transformation.
- Add your own code to the user-written code transformation.
- Add your own transformation to the Process Library by using the Transformation Generator.

Modify Configuration Files or SAS Start Commands

Several suggestions are made in this paper for invoking SAS with customizations to the environment. You might want to apply specific settings to a particular SAS program or to all jobs that run on a particular server(s). You can do this by adding parameters to the SAS startup command, modifying the `config.sas` file, or enhancing the `autoexec.sas` file.

The options -`UTILLOC` and -`NOWORKINT` are examples of the types of options that are used in the configuration file or as parameters in the SAS startup command. -`UTILLOC` and -`NOWORKINT` are invocation options and can only be set when you initialize a SAS session. These options cannot be included in a SAS program or used in a process flow that defines a job in SAS Data Integration Studio.

You can specify other options and settings for a SAS program in the `autoexec.sas` file. The SAS Workspace Server(s) must be set up to use your customized `autoexec.sas` file for jobs that you run in SAS Data Integration Studio. The `autoexec.sas` file can contain any valid SAS code. Some options that can be included are: `SORTSIZE`, `NONOTES`, and `SYMBOLGEN`. Options that are specified in the `autoexec.sas` file affect **all** programs that are run with that file, unless the program has code that overrides those options.

For more information about invocation options, `config.sas`, and `autoexec.sas`, see *SAS OnlineDoc 9.1.3* available at support.sas.com/onlinedoc/913/docMainpage.jsp.

MPRINT Option in SAS Data Integration Studio

Code that is generated by SAS Data Integration Studio contains conditional logic that turns on the `MPRINT` option at the start of every job, regardless of whether the option is set in the startup command for the server or in the configuration files. Turning on the `MPRINT` option sends additional information to the SAS log, which is essential for debugging SAS macro code. This can be an important tool when developing and testing your process flows.

After you have tested a process flow and deployed the job as a SAS program for production use, you might want to evaluate the need for using the `MPRINT` option. Having the option **On** for production runs is usually desirable because you might still have debugging issues. If there is a concern, such as SAS logs becoming too large, you might want to suppress the `MPRINT` lines. You can accomplish this **without having to edit the deployed program**. Add the following code to the `autoexec.sas` file for your production run:

```
%let etls_debug=0;
```

If the condition `etls_debug=0` is true, the logic in the deployed job prevents execution of the `OPTIONS MPRINT;` statement. To turn on the `MPRINT` option again, remove `%let etls_debug=0;` from the `autoexec.sas` file.

Caution: It is highly recommended that you do not turn off the `MPRINT` option in a development environment.

Specify Options on the Code Generation Tab

In SAS Data Integration Studio, you can specify options on the **`Code Generation`** tab in the general Options window to affect the code that is generated for all new jobs. This is where you can specify many of the settings that are used for parallel and grid computing.

To open the general Options window from the SAS Data Integration Studio desktop, on the toolbar, click **Tools ►Options** For a description of the available options, see the SAS online Help for the tab.

Add SAS Code on the Pre- and Post-Processing Tab

You can add SAS code on the **`Pre and Post Processing`** tab in the Properties window for an ETL flow. To open the Properties window, right-click the icon for the job in the **Inventory** tree and select **Properties** in the ETL flow pop-up window. Use this method to execute SAS code at the beginning or the end of a job. For more information about adding code, see the SAS online Help for the tab.

To re-direct the SAS log and SAS output to specific locations, add this code on the **Pre and Post Processing** tab:

```
PROC PRINTTO LOG='<filename>'

PRINT='<filename>';

RUN;
```

You can also specify OPTION statements for the following options: `FULLSTIMER`, `MSGLEVEL`, `SYMBOLGEN`, `UBUFNO`, `WORKTERM`, `BLKSIZE`, and `BUFSIZE` if you want these options to apply to an entire job.

SAS System Options for Transformations

You can specify SAS system options, SAS statement options, or transformation-specific options for many transformations on the **`Options`** tab or on other tabs in the Properties window.

To open the Properties window for a transformation in a job and specify options, follow these steps:

1. Open the job to display its ETL flow.
2. Right-click the transformation and select **Properties** from the pop-up menu.
3. In the Properties window, click the **`Options`** tab (or other tab as appropriate).

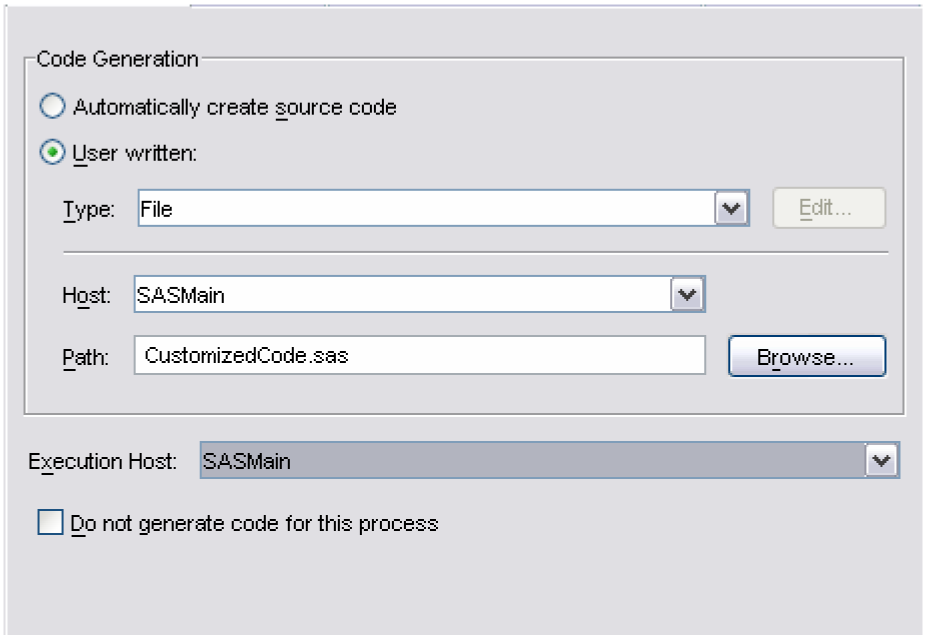
If the **`Options`** tab includes a system options field, you can specify options such as `UBUFNO` for the current transformation. Some transformations enable you to specify options that are specific to that transformation. For example, the **`Options`** tab for the SAS Sort transformation has specific fields for Sort Size and Sort Sequence. It also has a **PROC SORT Options** field in which you can specify sort-related options (which are described in the "Sorting Data" section) that are not surfaced in the interface.

In addition to the **`Options`** tab, some transformations have other tabs that enable you to specify options that affect performance. For example, the Properties window for the SAS Scalable Performance Server Loader transformation has an **`Other SPD Server`** **Settings** tab that enables you to specify multiple SAS Scalable Performance Server options.

For a description of the options that are available for a transformation, see the SAS online Help for the **`Options`** tab or for any tab that enables you to specify options.

Replace Generated Code for a Transformation with Customized Code

You can replace the generated code for a transformation with customized code. For example, you can edit the code that is generated for a transformation and save it so that the customized code will run whenever the transformation is executed. You can change back to have SAS Data Integration Studio automatically re-generate the step's code by clicking User written: on the `Process` tab (Figure 14). Column mappings between the edited step and the steps that precede or follow it will stay intact.


Code Generation

( ) Automatically create source code

(•) User written:

Type: File

Host: SASMain

Path: CustomizedCode.sas

Execution Host: SASMain

[ ] Do not generate code for this process

Figure 14. Process Tab for Setting Customized Code for a Transformation

You can replace generated code when a transformation does not provide all the choices that you need in its point-and-click interface and the provided option fields. For example, you can replace the generated code to change the default join order that is generated for the SQL Join transformation, as described in the section "SAS SQL Joins".

The following steps you through the process for customizing generated code. First, it is assumed that an ETL flow has already been created, and you know which modifications you want to make in the generated code for a specific transformation in the flow. It is also assumed that the flow is displayed in the Process Editor.

1. In the Process Editor, right-click the desired transformation and select **Process** ►**View Step Code**. Code is generated for the transformation and displays in the Source Editor window.
2. In the Source Editor window, edit the generated code to customize it.
3. When finished, click **X** in the upper-right corner of the Source Editor window.
4. Click **Yes** when asked if you want to save your changes. A file selection window displays.
5. In the file selection window, specify a path name for the edited code and click **Save**. The edited code is saved to a file.

After you save the customized code, set the transformation so that the customized code will run whenever the transformation is executed. To run the customized code, perform the following tasks:

1. In the Process Designer window, select the transformation and then click **File ►Properties** on the toolbar. The Properties window for the transformation opens.
2. Click the **`Process`** tab.
3. On the **`Process`** tab, click the **`User Written`** button. The **`Type`** field and related fields become active.
4. In the **`Type`** field, select **`File`**.
5. Usually, you will accept the default host in the **`Host`** field.
6. In the **`Path`** field, specify a path to the file that contains the edited code. The server in the **`Host`** field must be able to resolve this path. You can type this path in or click the **`Browse`** button to display a file-selection window.
7. After specifying the path, click **`OK`** to save your changes.

The specified user-written code is retrieved whenever code for this transformation is generated. From then on, the edited code is used in the job's code generation, until you re-select the option **Automatically create source code** on the **`Process`** tab. For more information about specifying user-written code for a transformation, see "Specifying User-Written Source Code for a Transformation in a Job" in the SAS Data Integration Studio online Help.

Add Your Own Code to User-Written Code Transformation

To create an ETL flow for a job, you can drag-and-drop transformations from the Process Library tree into the Process Editor. If the predefined transformations in the Process Library tree do not meet your needs for a task in a specific ETL flow, you can write a SAS program that will perform the desired task, add your own code to a User-Written Code transformation to the ETL flow, and then specify the location of the new code in the metadata for the User-Written Code transformation (Figure 15).

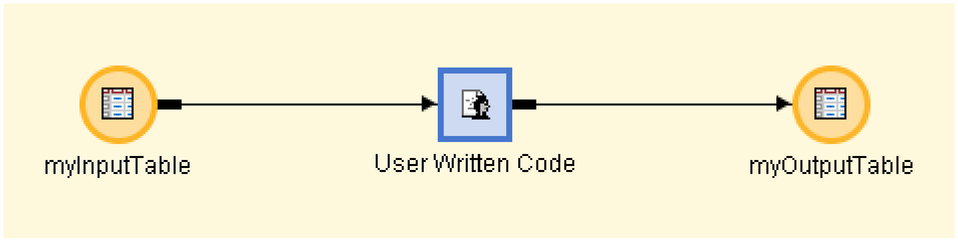


Figure 15. Adding Your Code to a User-Written Code Transformation in an ETL Flow

After the transformation is inserted, modify the code location so that it specifies the location of your user-written code transformation. This is done via the **`Process`** tab. When the ETL flow is generated, the user-written code is retrieved and inserted as part of the ETL flow (see Figure 15).

You can base your SAS program on code that is generated for one or more standard transformations in SAS Data Integration Studio, or you can use some other SAS code-generation tool such as SAS Enterprise Guide to help build the initial code.

The following steps you through the process of writing your own transformation. It is assumed that an ETL flow has already been created, that you have already written and tested the desired SAS program, and that you are ready to add the User-Written Code transformation to the flow. It is also assumed that the flow is displayed in the Process Editor.

1. Drag a User-Written Code transformation from the Process Library and drop it into the appropriate location in the ETL flow.
2. Right-click the icon for the User-Written Code transformation and select **Properties** from the pop-up menu.
3. On the **`Source`** tab, either paste in the revised code as metadata or specify the path to the saved code (if you will be maintaining user-written code in an external location).
4. Make any final edits of the code; be sure the input and output table names are correct. Specify `&SYSLAST` as your input table name if the input refers to the output from the preceding task. Specify `&_OUTPUT` as your output table name if your output is a single WORK data set that is being used to feed the next task.
5. If other tasks follow, manually define columns on the **`Mapping`** tab. Use the Quick Propagate option on the **`Mapping`** tab and apply additional edits to define the output columns correctly.

Tip: In an ETL flow, a User-Written Code transformation can have only one input and one output. If you need additional inputs or outputs, you can add a Generated Transformation (as described in the next section).

For more information about using a User-Written Code transformation in an ETL flow, see "Example: Include a User-Written Code Template in a Job" in the SAS Data Integration Studio online Help.

Add Your Own Transformation to the Process Library

If have custom code that you want to use in more than one place, or if there are multiple inputs or outputs for your custom code segment, use the Transformation Generator wizard to create a generated transformation.

To display the Transformation Generator wizard from the SAS Data Integration Studio desktop, click **Tools ►Transformation Generator** on the toolbar. In the wizard, specify the number of inputs and outputs, custom options (if any), then attach your user-written SAS code, just as you would for the User-Written transformation that is described in the preceding section. The newly-created transformation appears in the Process Library tree and is available for use in any SAS Data Integration Studio job.

Drop the generated transformation into the ETL flow and specify options. When the the ETL flow is generated, the user-written code is retrieved and included as part of the flow. For example, you can write a generated transformation that includes a program to call the CLEAR_WORK macro (see "Deleting Intermediate Files") in order to delete the intermediate files in a job and recover disk space.

Appendix 2: A Sample Scenario for Using Parallel ETL Processing

Using the list of ETL planning tasks from the section, "How to Approach a Parallel ETL Effort", a star schema data mart was built "from scratch" using SAS Data Integration Studio from operational data that is stored in text files. The newly created star schema dimensions and fact table were loaded in parallel into SAS SPD Server.

The usual execution cycle when building a star schema is to first build the dimension tables and index the tables. The second step is to build the fact table and link it to each dimension table via a lookup function. There are multiple dimension tables to be built so it was possible to execute each of the dimension builds in parallel. This allowed some processes to run concurrently in order to reduce execution time. In the sample case, there were five dimension tables. Therefore, up to five processes ran concurrently for the first phase of the star schema build.

The second phase in building the star schema focused on the longer running task (fact table build with dimension lookups) to see how that phase could be implemented in parallel. In the scenario, the fact table data was composed of multiple years of retail transaction data, therefore, it made sense to attempt to divide (partition) the data by time. The incoming transaction data was stored in text files by week, so Week became the natural partition on which to parallelize the fact table load process. Each process in the second phase used a week’s worth of data, performed dimension table lookups for each row, and loaded the data into the final fact table that was stored in SAS SPD Server. After all the weeks were processed into the fact table, the SAS SPD Server dynamic cluster capability was used to instantly merge all the independent week partitions into one, large fact table (for information about SAS SPD Server and the dynamic cluster capability, see the *SAS Scalable Performance Data Server 4.4: User's Guide* at support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/scalable_ug_9722.pdf). After the cluster snap was completed, the star schema build-out was completed.

Figure 16 shows the flow diagrams for the SAS Data Integration Studio jobs that executed phase II of the star schema build mentioned earlier (the dimension table lookup and fact table load phase). The master job in Figure 17 is a Loop transformation that calls the dimension lookup job (also shown in Figure 16). The Loop transformation is set up to assign one week’s transaction data to the sub-job, and execute it across the available resources via Platform LSF software. As input, the Loop transformation had a list of the weekly data files. For more information about the Loop transformation, see the sections "Loop Transformation Guide", "Loop Properties", and "Loop Options tab" in the *SAS® Data Integration Studio 3.3: User's Guide* avaliable at support.sas.com/documentation/onlinedoc/etls/usage33.pdf.

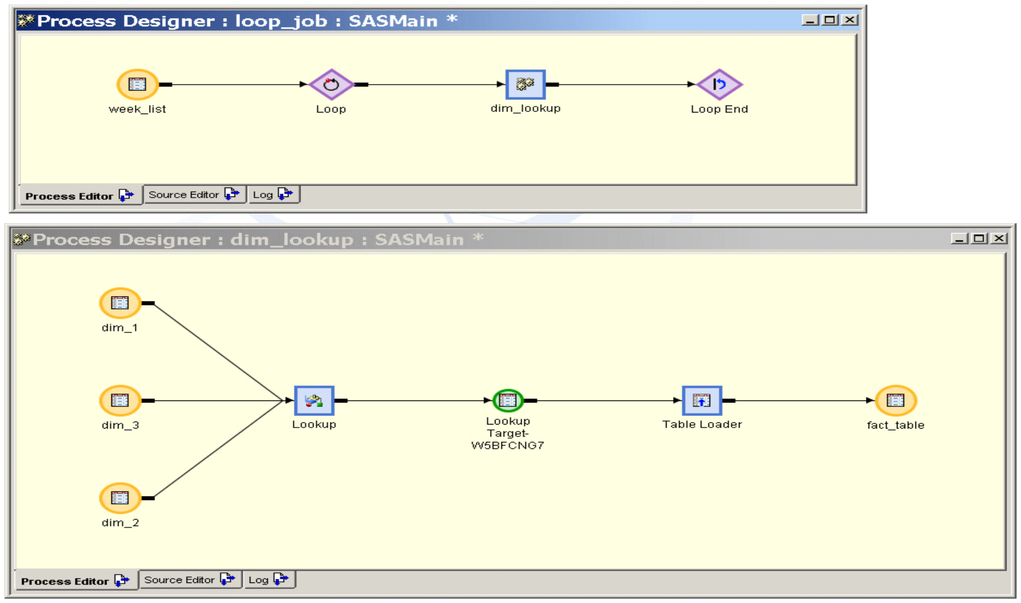
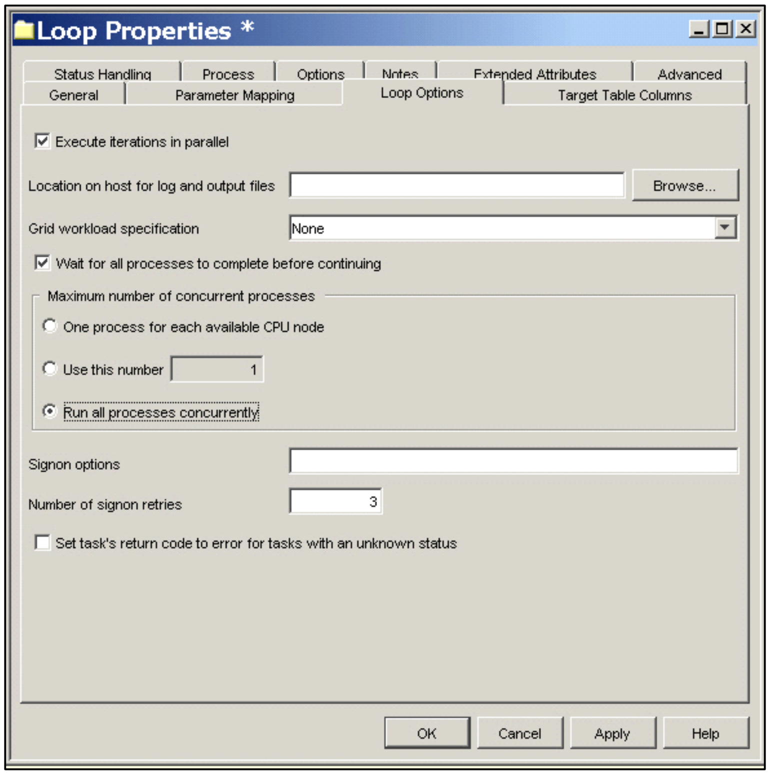


Figure 16. SAS Data Integration Studio Dimension Lookup

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.



[✓] Execute iterations in parallel

Location on host for log and output files

Grid workload specification: None

[✓] Wait for all processes to complete before continuing

Maximum number of concurrent processes

- ○ One process for each available CPU node
- ○ Use this number 1
- ● Run all processes concurrently

Signon options

Number of signon retries: 3

[ ] Set task's return code to error for tasks with an unknown status

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.

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.

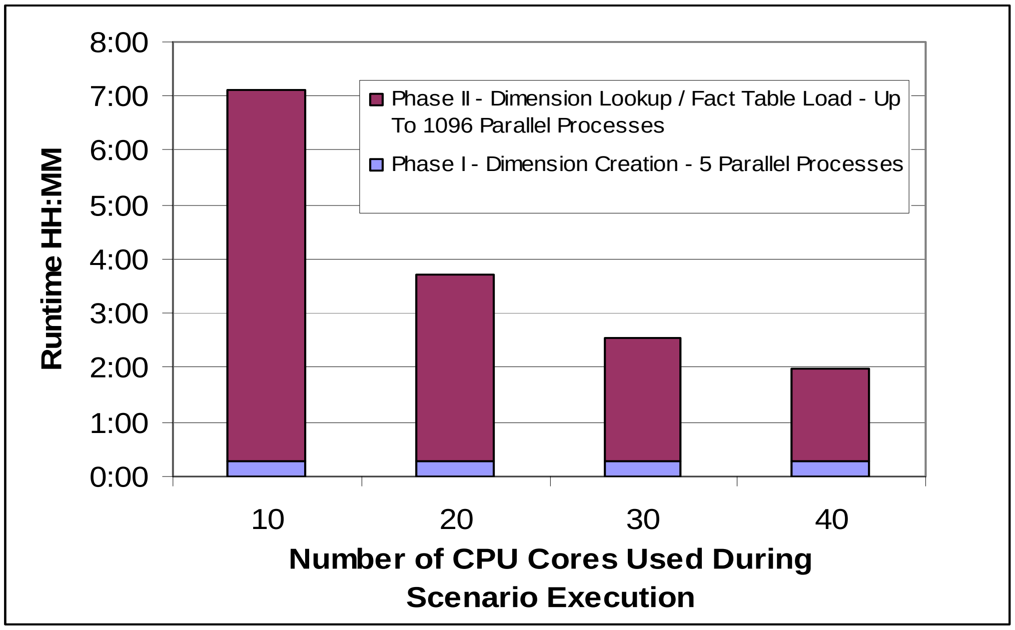


Figure 18. Results from Running the Star Schema Parallel Build across Multiple CPU Cores

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 <u>any</u> 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. 2005. "Setting Up and Tuning Direct I/O for SAS®9 on AIX". Raleigh, NC: IBM Corporation Inc. Available at www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100890.

Karp, Andrew H. and David Shamlin. 2003. "Indexing and Compressing SAS® Data Sets: How, Why, and Why Not". *Proceedings* of the 28th Annual SUGI Conference. Cary, NC: SAS Institute Inc. Available at 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 25th Annual SUGI Conference*. Cary, NC: SAS Institute Inc. Available at 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.

SAS Institute Inc. 2007. SAS® Install Center Web Site. SAS® 9.1.3 Intelligence Platform: Administration Documentation. Available at support.sas.com/documentation/configuration/913admin.html.

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.

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

SAS Institute Inc. 2006. *SAS OnlineDoc® 9.1.3.* Available at 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.

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

Shoemaker, Jack. 2001. "*Eight PROC FORMAT Gems*". *Proceedings* of the 26th Annual SUGI Conference. Cary, NC: SAS Institute Inc. Available at 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**