Paper 108-2007

Super Size It!!! Maximize the Performance of Your ETL Processes

Nancy A. Rausch and Nancy J. Wills, SAS Institute Inc., Cary, NC

ABSTRACT

The success of every business activity—from supplier management to customer service—is dependent upon how well an organization manages its critical data. This paper discusses practical recommendations for optimizing the performance of your Extract, Transform, and Load (ETL) data management processes. The paper presents best practices in ETL process development and includes performance, tuning, and capacity information. You will learn to analyze and debug ETL flows, gain efficiency quickly, optimize your system environment for performance, and customize with advanced performance techniques.

INTRODUCTION

Building efficient processes to extract data from operational systems, transform the data into the appropriate data structures that form the data warehouse, and load 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 paper discusses two specific areas used in most ETL processes: joins and loading. The paper discusses techniques for improving performance in those areas. These techniques help ensure that the processes are efficient, perform well, and scale well when data sizes increase.

JOINS

Joins are a common and resource-intensive part of ETL. SAS SQL implements three well-known join algorithms: sort-merge, index, and hash. This paper introduces some common techniques to aid join performance, along with some options that can influence which join algorithm the SAS SQL optimizer chooses. We also look at some optimizations you can do for multi-way joins and joins involving relational database tables.

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. The following best practices should help you optimize performance of your joins.

- *Presort Data for Joins*
  Presorting 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 presorting. Presorting might not be needed if you have large amounts of data, and you are subsetting the data for the join.

- *Optimize Order of Columns*
  Columns should be ordered in the SELECT statement as follows:
  - noncharacter (numeric, date, datetime) columns first
  - character columns last

  In a join context, all noncharacter SAS columns consume 8 bytes. Ordering noncharacter 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.

  In SAS® Data Integration Studio, use a transform's **Mapping** tab to rearrange columns by column type. Do this by sorting the "type" column in the target table in descending order.

- *Drop 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.

There are options that you can use to influence the SAS SQL optimizer to use the join algorithm that you prefer. These options can only **influence** the optimizer; they cannot force it.

First though, you need to understand 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. Here are 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
```

Now let's look at the different types of join algorithms SAS SQL can use and the options used to influence the join algorithm.

SORT-MERGE JOIN

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. For more information on sorting, see the "Recommended Reading" section at the end of this paper.

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 transform (Figure 1) to add MAGIC=102 to the PROC SQL invocation.

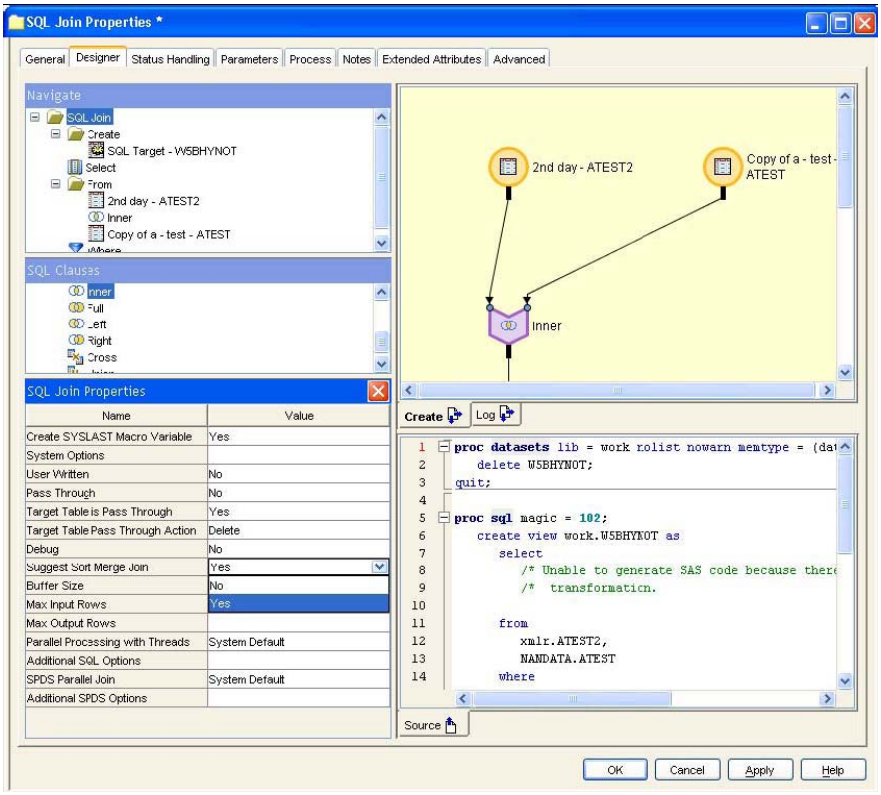

Figure 1. Sort-Merge Join Property Turned On in Version 2 of the SQL Join Transform

INDEX JOIN

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.

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

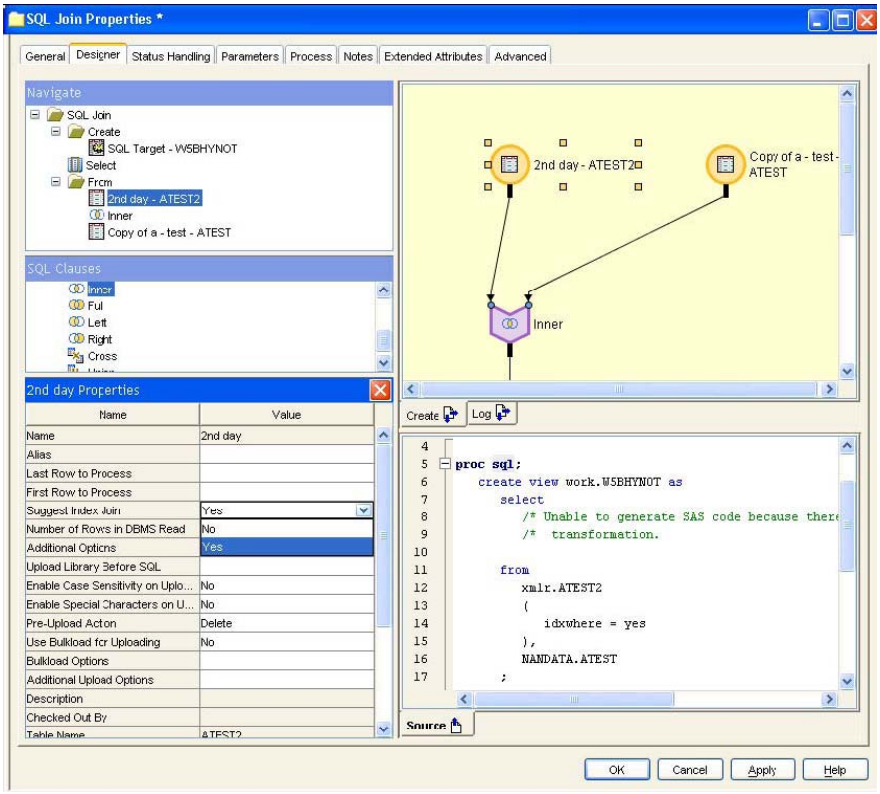

Figure 2. Suggest Index Join Property Turned On in Version 2 of the SQL Join Transform

HASH JOIN

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

An internal memory-sizing formula determines whether or not a hash join is chosen. The formula is based on the PROC SQL option BUFFERSIZE=, whose default value is 64K. If you have enough memory on the hardware you are using, consider increasing the value for BUFFERSIZE= in order to increase the likelihood that a hash join is chosen (Figure 3).

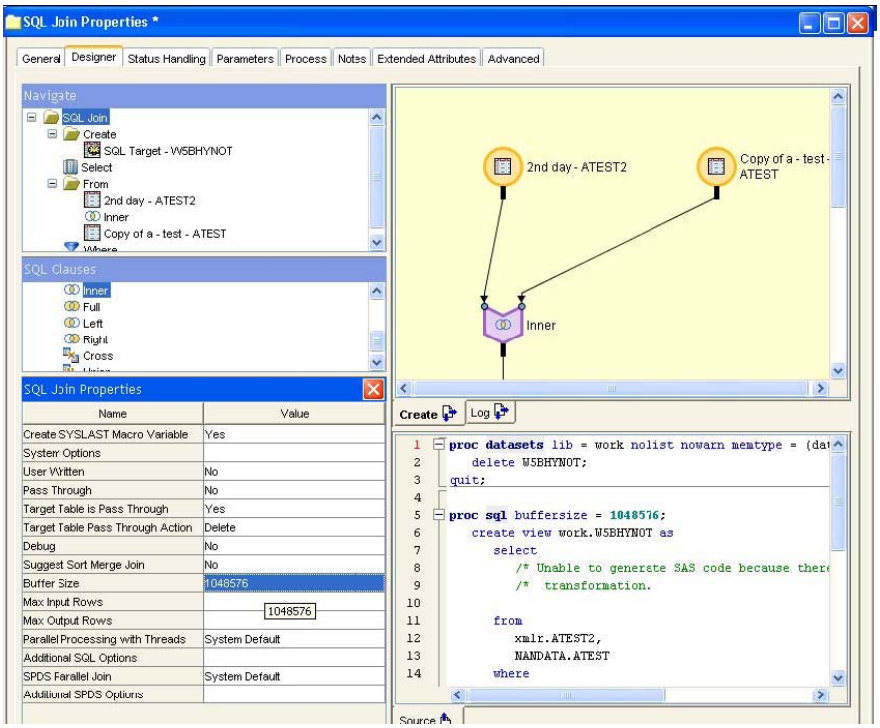

Figure 3. Changing Buffer Size Property in Version 2 of the SQL Join Transform

UNDERSTANDING MULTI-WAY JOINS

Many joins are two-table joins but other joins are multi-way joins involving more than two tables. To understand how to optimize joins, you also need to understand *multi-way joins*. Multi-way joins 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.

Because of the number of temporary tables that are required to execute a multi-way join, these joins can often run out of disk space. This is because SAS SQL does not release temporary tables (except sort utility files), 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. To avoid space problems, you can either make sure there is enough space for SAS WORK to handle the joins or delete the temporary tables between join steps.

Many multi-way joins involve inner joins. SAS SQL allows you to code inner joins using two different forms of SAS SQL syntax. The syntax you use can influence the performance of your inner joins.

By nature, the SAS SQL inner join operation allows the SAS SQL optimizer to rearrange 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 rearrange 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.

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.

SAS SCALABLE PERFORMANCE DATA SERVER STAR JOINS

You can use the SAS Data Integration Studio SQL Join transform 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 containing two or more dimension tables that surround a centralized fact table. This 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 pairwise 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 in the library definition:

- 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:

```
WHERE 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
```

You can use the **Where** tab of the SQL Join transform (see bottom of right panel on the **Designer** tab in the SQL Join Properties window) to build the condition that is expressed in the preceding WHERE clause. Figure 4 shows how the **Where** tab depicts the preceding WHERE clause.

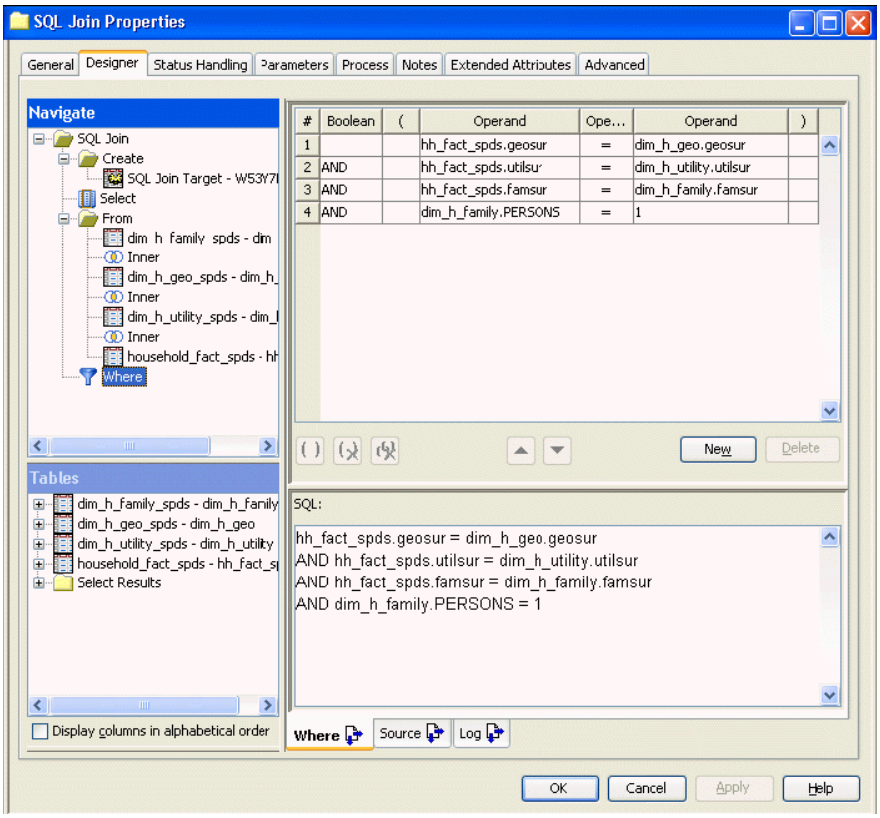

Figure 4. The Where Tab in Version 2 of the SQL Join Transform Showing a WHERE Clause That Will Enable an SPD Server Star Join

Note: The SAS Scalable Performance Data Server requires all subsetting to be implemented on the **Where** tab in the SQL Join transform.

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* (SAS Institute Inc. 2006).

RELATIONAL DATABASE JOINS

So far, we have talked about joining multiple SAS data sets or SPD Server data. Now let's look at joins that include tables from relational databases.

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 tables 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 relational 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. SAS rules and algorithms 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. If this happens, you may want to influence SAS SQL to use another algorithm.

Let's take a closer look at how SAS SQL passes SQL to the relational database. This will help you understand how to write SAS SQL that will pass to the database, and will help you understand why your SAS SQL did not generate database SQL. This information applies to all SAS SQL queries, and is not limited to join statements.

Because PROC SQL operates on SQL, it can readily provide more efficient mechanisms to pass SQL to a database. SAS SQL supports two methods that allow passing more complex SQL queries through a SAS engine to an underlying database. These methods are:

- The SQL Pass Through facility (Explicit Pass-Through)
- SQL Implicit Pass-Through (Implicit Pass-Through)

Explicit Pass-Through offers you complete control of the SQL syntax sent to a database, but it requires a query to be written in the SQL syntax of the database.

Implicit Pass-Through is an optimization technique in PROC SQL for queries using a single SAS/ACCESS® engine. SQL Implicit Pass-Through attempts to reconstruct the textual representation of a query in database syntax, from PROC SQL's internal SQL tree representation. If successfully reconstructed, the textualized query is then passed to the database.

Although Implicit Pass-Through cannot support passing the unlimited variety of SQL that is possible with Explicit Pass-Through, there are many benefits to using Implicit Pass-Through. Implicit Pass-Through allows:

- SAS procedures to access relational database data directly
- You to write SAS program code that operates on many data sources without requiring a detailed understanding of the underlying data
- You to easily perform operations that combine heterogeneous (divergent) data sources
- SAS analytics to be applied directly to database data as easily as native SAS data stores

SAS/ACCESS engines have incorporated many features to improve performance, including buffering of the data read from the database, parallel threading of database queries, exploiting a database's bulk-loading facilities, methods for subsetting the data to minimize transfer, support of temporary tables within the DBMS, and techniques for performing 'indexed' type query operations.

Although there is no guarantee that a given SQL query can be passed to a database, the Implicit Pass-Through SQL optimization will try to do everything it can to generate SQL that will pass to the database. There are various reasons why the Implicit Pass-Through optimization technique might not succeed. Single- or multiple-table homogeneous queries that contain one of the following constructs will cause an Implicit Pass-Through processing attempt.

- Select DISTINCT keyword
- Query contains an SQL Aggregate function
- Query contains a GROUP BY clause
- Query contains a HAVING clause
- Query contains an ORDER BY clause
- Query is an SQL join
- Query involves a Set operation, other than an OUTER UNION
- Query with a WHERE clause containing a Subquery

Whenever the SQL Implicit Pass-Through optimization is unsuccessful, PROC SQL will continue to process the query internally. The query is executed, but the query's performance might not be optimal. There are certain SQL constructs that automatically disqualify a query from Implicit Pass-Through processing. The following constructs inhibit Implicit Pass-Through processing. For performance-critical SQL steps including joins, avoid these constructs if possible.

- Heterogeneous queries
- Queries that incorporate Explicit Pass-Through statements
- Queries that use SAS data set options
- Queries that contain the SAS OUTER UNION operator
- Specification of a SAS Language function that isn't mapped to a database equivalent by the engine
- Queries that require remerging in PROC SQL
- Specification of ANSIMISS or NOMISS keywords in the SQL join syntax
- The SAS/ACCESS DIRECT_SQL= LIBNAME statement option
- Using SAS/ACCESS LIBNAME statement options that indicate member level controls are desired

We've looked at what you should and should not do to ensure that your SAS SQL is passed down to the relational database. What if you must join heterogeneous data sources?

The ability to easily perform join operations that involve heterogeneous data sources is a powerful feature of SAS SQL. However when a PROC SQL query references different data sources—such as with a join between SAS and database tables—the entire PROC SQL query cannot be passed to the database by the SQL Implicit Pass-Through facility. One or more of the joins are performed in SAS by PROC SQL.

Standard heterogeneous joins can perform poorly if very large database data is read into SAS where the join is then performed.

For performance-critical code steps or cases when a join might be executed more than once, there are three choices that might increase performance:

- Copy database data into SAS data sets, and join SAS data sets.
- Copy SAS data to a database table and perform the join in the database. You can choose to use temporary table space in the relational database management system (RDBMS) to hold the data.
- Enhance join performance in SAS by sorting on the join key.

Copying database data to SAS data sets is similar to performing a standard heterogeneous join: a large volume of database data might be transferred to SAS. In most cases this approach is worthwhile only when multiple joins reference the database data. By copying to a SAS data set, you incur the data transfer cost only once. You change your joins to reference SAS copies of the tables, and your joins become SAS-to-SAS joins.

SAS Data Integration Studio generates explicit pass-through code and supports many optimizations for optimizing the performance of your queries when working with databases. These optimizations assist in getting the greatest performance out of the data because they deliver the source data to the database prior to the join and make sure the target data remains at the target. Moving data between SAS and/or the database can be costly in terms of performance, so it is best to upload the data to the database if your join can be completed at the database. Once the data is in the database, the data can be efficiently joined by using either implicit SQL or explicit SQL.

The options available in SAS Data Integration Studio for optimizing joins and target table data include (Figure 5):

- Enable/disable explicit pass-through
- Specify if the target table is pass-through
- Specify if the target table should be dropped and recreated when loading, or truncated (cleared of records) when loading
- Whether or not to use the Bulk Loader when loading the target table

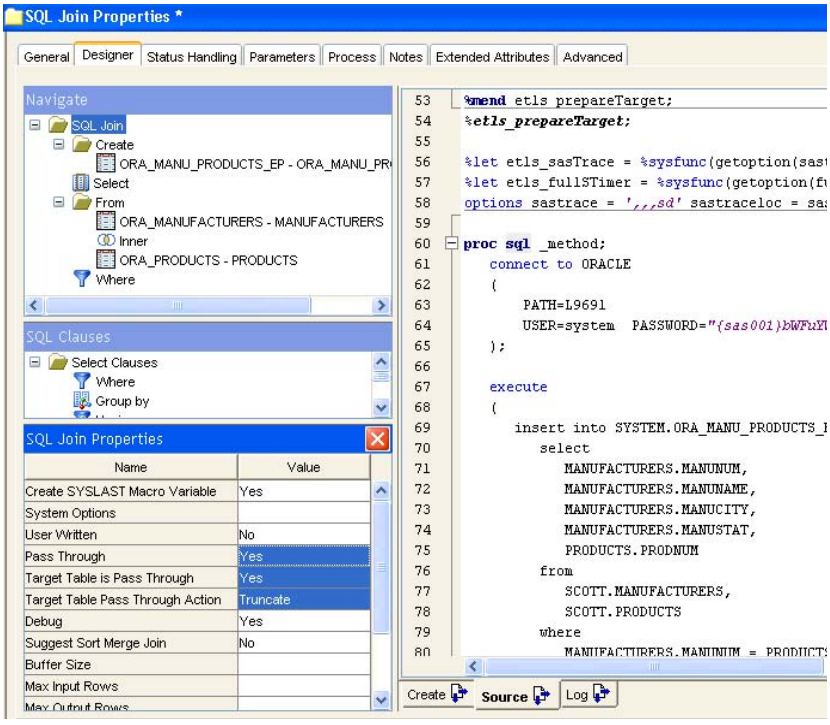

Figure 5. Explicit Pass-Through Optimizations Available in Version 2 of the SQL Join Transform in SAS Data Integration Studio

Additional options are available to assist in optimizing the source data being joined. These optimizations assist in getting the greatest performance out of the data because they deliver the source data to the database prior to the join. Once the data is at the database, the data can be efficiently joined by using either implicit SQL or explicit SQL.

The options available for each source table participating in the join the SQL Join Transform in SAS Data Integration Studio include (Figure 6):

- Indicate if the source table is to be uploaded to the database before the join.
- Specify if the source table has special characters when uploading.
- Specify the technique to use for the upload and the optional library to hold the temporary table prior to the join.
- Delete or truncate the table in the temporary source library prior to loading with the new data.
- Determine whether or not to use the bulk loader to upload the data.
- Indicate if there are additional options associated with uploading the source data.

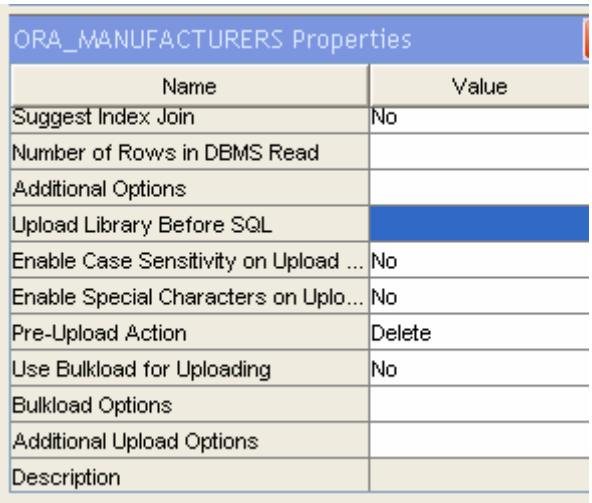

Figure 6. Optimizations for Getting Source Data to the Database in Version 2 of the SQL Join Transform in SAS Data Integration Studio

LOADING DATA

The final step of an ETL process involves loading data into permanent physical tables. The performance of your ETL processes can be significantly impacted by this data load. A number of factors affect load performance including the format of the table, its structure and location, whether it has constraints and indexes, and how much data will be loaded. As a designer or builder of the ETL process flow, you want to select the best load technique for your data.

There are three basic types of load techniques (called load styles) that ETL designers typically use when loading tables:

- Add incoming records to the target table
- Replace/refresh the table
- Update records in the target based on changes coming from the source.

Sometimes more than one technique is required to correctly perform a load. For example, there might be new and updated records that have to be loaded into the target table. In addition, different techniques might be required for the same table at different times. In a retail data model, you can do daily updates to collect store sales for example. Then once a week, you can roll off the daily information to a longer-term storage table, drop all of the records, and recreate the tables in preparation for the next week. Different scenarios might require a different load technique for a table in order to get the best performance. Therefore, the ETL process should be optimized to match the type of load that has to be performed on the table.

It is important to also consider that many relational database systems supply various features to help optimize load performance based on the type of load required. For example, some relational database vendors support BULKLOAD facilities, which are very efficient when loading large amounts of data into tables. Others support optimized techniques for refreshing tables and updates that can also efficiently handle adding records. When using a relational database to store data, these load techniques might frequently offer the best load performance.

TABLE LOADER TRANSFORM

SAS Data Integration Studio 3.4 provides a high-performing Table Loader transform. It generates the code using the SAS/ACCESS engines and Base SAS® engines technologies to optimize the loads for performance. The Loader Transform selects an optimum default load style for you based on the storage type for your data (relational database or SAS) and provides many customizations to help you tailor the load style to meet your needs.

The Table Loader transform supports the ability to perform any of the three load styles, as shown in Figure 7. The transform generates the code required to load SAS data sets, database tables, and other types of data, such as an Excel spreadsheet. When loading tables, the transform can maintain indexes and constraints on the table that is being loaded.

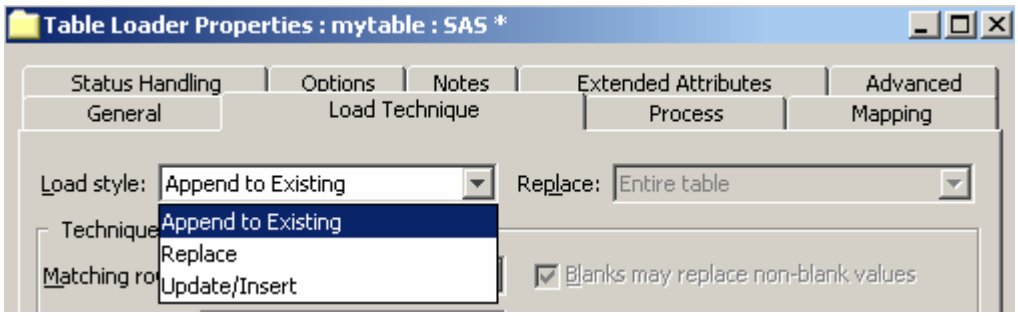

Figure 7. Load Styles on the Table Loader Transform Load Technique Tab

Depending upon 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 transform generates code to do a combination of the following tasks:

- Remove all rows
- Add new rows
- Match rows and 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 method is called when you choose the Replace load style. Replace is the load style to choose when you want to recreate the table. You can choose to completely delete the table and then recreate it, or keep the original table structure, remove all records, and then add the incoming records. This technique is frequently used for initial loads of data tables, or when you don't need to retain history. It can also be used for intermediate tables and for tables that result from a query in which the results don't need to be maintained.

There are three types of options that can be used to remove all rows from a table. These options are associated with the Replace load style that is available in the Table Loader Transform:

- 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 (available for only some databases)
- Replace all rows using delete—uses PROC SQL with DELETE * to remove all rows.

To completely delete the table and recreate it, choose the option "Replace entire table." When you select this option, the table is deleted and then recreated with zero rows. This technique is normally used when you don't have any restrictions on your ability to be able to recreate the table and is frequently used for SAS data sets. It is not normally used in relational database tables. This technique is not suitable when your security requirements restrict table deletion permissions or when there are table space limitations, which are restrictions commonly imposed by a database administrator on database tables. You might also want to avoid this method if the table has any indexes or constraints that are not easily recreated from metadata stored on the table. Check constraints and column range constraints are two examples of this type of constraint that would not be recreated when the table is created.

The option "Replace all rows using truncate" is available only when your target table resides in one of the relational databases that support the TRUNCATE feature. It is the best choice for database tables. This is because this option generates passthru code specific for the relational database to perform the TRUNCATE in the database directly, which is optimized by the database for performance. For these types of tables, this is the fastest method available to clear the table. Check with your RDBMS supplier to see if your database supports this feature.

If your relational database system does not support the TRUNCATE feature, the “Replace all rows using delete” is the next best choice. The "delete *" syntax required to remove the rows will work but—depending on database and table settings—might incur some overhead that can make it slower than either of the other two techniques. Note that either of the "Remove all rows..." choices allows you to leave all indexes and constraints intact during the load.

You can also use “Replace all rows using delete” for SAS tables when you don’t want to completely delete the table, since it allows indexes and constraints to remain on in the table. This does have one drawback, however. For SAS tables, the actual size of the SAS data set is not truncated each time the delete runs. This is because SAS data sets support only a logical delete of their records. Unless the table is deleted and recreated, its physical size continues to grow and, over time, the increased size can negatively affect performance. For this reason, if you choose the “Replace all rows using delete” technique for a SAS table, you should monitor the size of the underlying data set over time, to make sure it isn’t getting too large.

ADD NEW ROWS

This method is called from all three load styles when there are new records to be added to the target table. There are two choices available to you to configure the way in which the add is performed for any of the three load styles:

- PROC APPEND with the FORCE option
- PROC SQL using the INSERT statement

PROC APPEND with the FORCE option is the default choice. If the source is a large table and the target is in a database that supports bulk load, PROC APPEND takes advantage of the bulk load feature; see the section "Bulk Loading into Relational Databases". The second technique, PROC SQL using the INSERT statement, will perform well when the table that is used for appending is small. This is because the setup for a bulk load has some overhead associated with it when it runs, and PROC SQL using the INSERT statement inserts one row at a time into the target table. This makes it suitable for small tables, but not for larger tables.

MATCH ROWS AND UPDATE ROWS

This method is called when you select the Update/Insert load style. You select Update/Insert when you have incoming data that will be used to update data in the target table. This is probably the most common data warehousing load style for tables because it is the technique that supports updating information in the warehouse with the incoming records. Update/Insert will support updating existing records, and adding new records to the table, but does not support deleting any records out of the tables.

When you select Update/Insert, you have to select which columns or index to use to match the incoming records to so that the loader can perform the Update. The Table Loader offers several ways to match:

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

Each of these will update matching rows in the target. The MODIFY BY and MODIFY KEY= techniques have the added benefit of being able to take unmatched records and add them to the target during the same pass-through of the source table.

The MODIFY KEY= technique often outperforms the other update methods when loading SAS tables. Note that to use this technique an index on the table is required. MODIFY KEY= can perform adequately for database tables as well when indexes are used. Performance of the PROC SQL SET or MODIFY BY techniques will vary. Neither of these techniques requires data to be indexed or sorted, though indexing on the key column(s) can greatly improve performance. Both use WHERE processing to match each row of the source table with one in the target.

These techniques each perform differently based on the percentage of rows being updated. If the majority of target records are being updated, the DATA step MERGE (or UPDATE) might perform better than the DATA step MODIFY (or PROC SQL). This is because MERGE makes full use of record buffers. Performance results can be hardware- and operating system-dependent, so consider testing more than one technique.

INDEXES AND CONSTRAINTS CONSIDERATIONS

In some situations, removing nonessential indexes before a load and re-creating those indexes after a 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 reloaded. When you leave the indexes on, the indexes are built continually as you add records. This overhead can get excessive as the volume of data being loaded increases.

You might want to try temporarily removing key constraints in order to improve performance. If there is 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 8) of the Table Loader transform. There are four settings that enable you to specify the desired conditions for the indexes and constraints before and after the load.

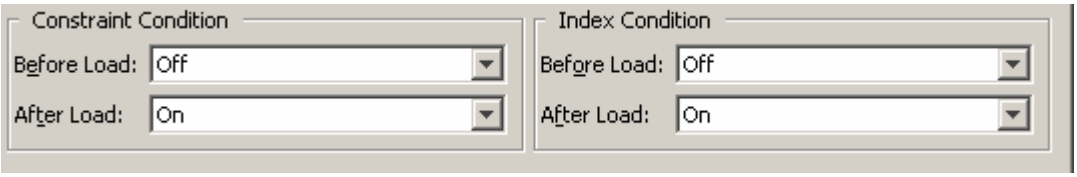

Figure 8. Constraint and Condition Options on Load Technique Tab of Table Loader Transform

The options that are available will depend on the load technique that you choose. The choices essentially provide the ability to

- put constraints and/or indexes on
- take constraints and/or indexes off
- leave constraints and/or indexes as they were before the load

When you select OFF for the "Before Load" option, the generated code checks for and removes any indexes or constraints on the table, 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 to have indexes added back after the load. If you decide not to build indexes after the load, your physical table will be out of sync with the metadata for the table, which is normally not desirable.

One scenario where you might want to leave indexes off during and after loading for performance reasons is when indexes have been defined only on the table to improve the performance of a query. Another scenario could be if the table is being updated multiple times in a series of different load steps that appear in separate jobs. None of the load steps need indexes; therefore, leaving indexes on impedes the performance of the load. In this scenario, indexes can be taken off before the first update and left off until after the final update.

BULK LOADING INTO RELATIONAL DATABASES

One of the fastest ways to insert large data volumes into a relational database when using 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 LIBNAME option (Figure 9), a database bulk load method is used. This can significantly enhance performance, especially when database tables are indexed. Each SAS/ACCESS engine invokes a different loader and uses different options. Some SAS/ACCESS engines support multiple loaders.

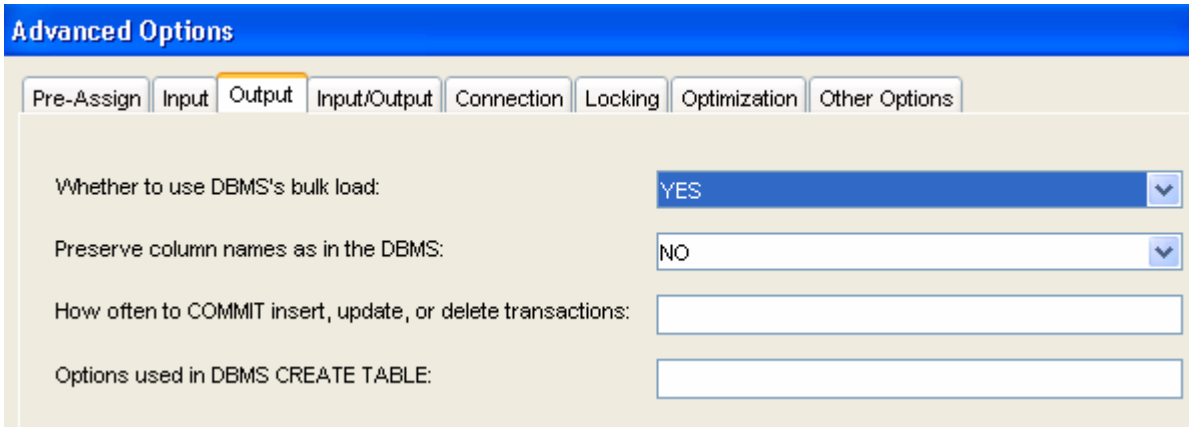

Figure 9. Library Option to Enable BULKLOAD for a Database

The primary advantage of bulk load is the speed of the load, especially on database tables that have indexes defined. The difference in processing time between a bulk load and a standard insert can be significant.

Bulk loading of indexed database tables is performance-optimal, especially when appending many rows to a large indexed database table. When you use a bulk loader, both data and index table components are loaded in bulk. Thus, you avoid the costly standard insert alternatives of dropping and recreating indexes or database index updates on each row that is inserted.

Using a database bulk loader incurs some additional overhead. A bulk loader engages in a special multiphase protocol with the database server. The protocol phases take time. By comparison, a standard insert is relatively overhead-free; therefore, on very small numbers of rows, a standard insert outperforms a bulk load.

Bulk load reduces load time but introduces complexity; for example:

- Additional database-vendor-supplied software might be required.
- SAS might need to be configured to allow the X command.
- Database-enforced referential integrity (RI) might inhibit bulk load.
- Database indexes might inhibit bulk load for Teradata
- Deferred error detection can cause unexpected behavior.
- Error correction might be difficult.
- The ERRLIMIT option might not be honored.

If bulk load is not the right process for your job but you still are concerned about performance, consider using the INSERTBUFF option. The INSERTBUFF option specifies the number of rows that can be held in memory for inserting into the database from SAS. The INSERTBUFF option can be specified as a data set option or as an option in the LIBNAME statement. You might need to experiment with different values for this option to determine the optimal value for your application. Optimal performance will vary with network type, network traffic, and available memory in SAS and the database.

In the SQL Transform in SAS Data Integration Studio, you can specify BULKLOAD, INSERTBUFF, and other options to control where and how the join will occur (Figure 10). The source tables can be uploaded to the target database prior to the join using BULKLOAD options available in the SQL transform.

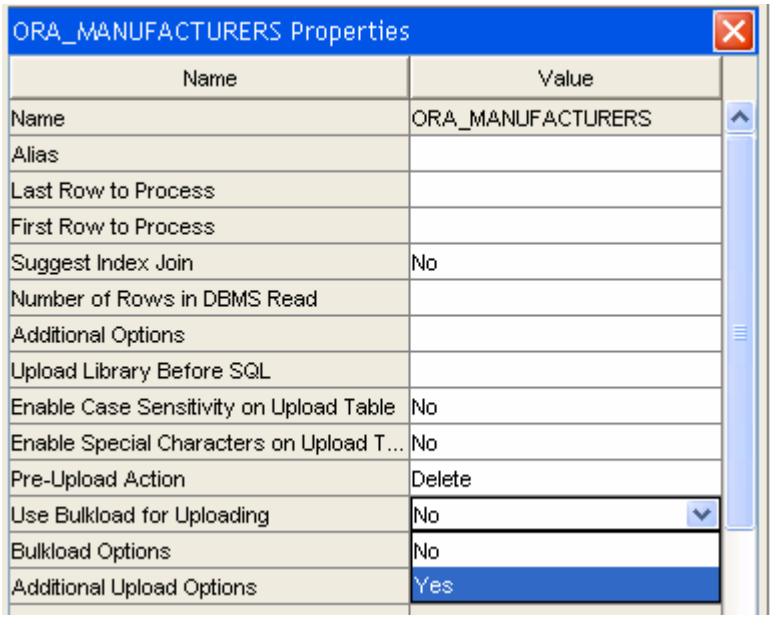

Figure 10. SQL Transform in SAS Data Integration Studio Supports BULKLOAD and Other Options to Assist in Optimizing the Performance of Your Join

CONCLUSION

In this paper, we just looked at ways to improve performance while joining data and loading data. Last year at the Thirty-First SAS Users Group International Conference (SUGI 31), we presented "Practical Tips for Increasing the Performance of your SAS Processes". This year, we are pleased to offer a white paper, "ETL Performance Tuning Tips" (SAS 2007) that contains more extensive information on the topics presented in this paper, plus many other topics related to building ETL flows using SAS Data Integration Studio.

RECOMMENDED READING

SAS Institute Inc. 2007. "ETL Performance Tuning Tips." Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/whitepaper/technical/ETLperformance07.pdf.

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

SAS Institute Inc. 2006. *SAS® Scalable Performance Data Server® 4.4: User's Guide, Second Edition.* Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/scalable_ug_9722.pdf.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Nancy Rausch
SAS Institute Inc.
Cary, NC 27513
nancy.rausch@sas.com

Nancy Wills
SAS Institute Inc.
Cary, NC 27513
nancy.wills@sas.com