Adding a Little Magic to Your Joins

Kirk Paul Lafler, @sasNerd, Spring Valley, California

ABSTRACT

To achieve the best possible performance when joining two or more tables in the SQL procedure, a few considerations should be kept in mind. This presentation explores options that can be used to influence the type of join algorithm selected (i.e., step-loop, sort-merge, index, and hash) by the optimizer. Attendees learn how to add a little magic with MAGIC=101, MAGIC=102, MAGIC=103, IDXWHERE=Yes, and BUFFERSIZE= options to influence the SQL optimizer to achieve the best possible performance when joining tables.

INTRODUCTION

The SQL procedure is a simple and flexible tool for joining tables of data together. Certainly many of the join techniques can be accomplished using other methods, but the simplicity and flexibility found in the SQL procedure makes it especially interesting. This paper presents several options MAGIC=101, MAGIC=102, MAGIC=103, IDXWHERE=Yes, and BUFFERSIZE= to influence the SQL optimizer in selecting the most efficient join algorithm possible.

WHY JOIN ANYWAY?

As relational database systems continue to grow in popularity, the need to access normalized data stored in separate tables becomes increasingly important. By relating matching values in key columns in one table with key columns in two or more tables, information can be retrieved as if the data were stored in one huge file. Consequently, the process of joining data from two or more tables can provide new and exciting insights into data relationships.

EXAMPLE TABLES

A relational database is simply a collection of tables. Each table contains one or more columns and one or more rows of data. The examples presented in this paper apply an example database consisting of three tables: CUSTOMERS, MOVIES, and ACTORS. Each table appears below.

CUSTOMERS

CUST_NO	NAME	CITY	STATE
11321	John Smith	Miami	FL
44555	Alice Jones	Baltimore	MD
21713	Ryan Adams	Atlanta	GA

MOVIES

CUST_NO	MOVIE_ID	RATING	CATEGORY
44555	1011	PG-13	Adventure
21713	3090	G	Comedy
44555	2198	G	Comedy
37753	4456	PG	Suspense

ACTORS

MOVIE_ID	LEADING_ACTOR
1011	Mel Gibson
2198	Clint Eastwood
3090	Sylvester Stallone

PROC SQL JOIN ALGORITHMS

Not one, but several PROC SQL join algorithms are available to the SQL optimizer. Based on the constructed query and the underlying table structures, the SQL optimizer determines which of the four available join algorithms to use when executing the join query. The available join algorithms are:

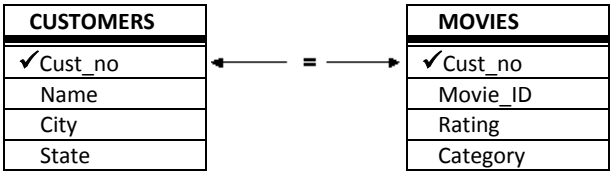
- ✓ **Nested Loop** – A nested loop join algorithm may be selected by the SQL optimizer when processing small tables of data where one table is considerably smaller than the other table, the join condition does not contain an equality condition, first row matching is optimized, or using a sort-merge or hash join has been eliminated.
- ✓ **Sort-Merge** – A sort-merge join algorithm may be selected by the SQL optimizer when the tables are small to medium size and an index or hash join algorithm have been eliminated from consideration.
- ✓ **Index** – An index join algorithm may be selected by the SQL optimizer when indexes created on each of the columns participating in the join relationship will improve performance.
- ✓ **Hash** – A hash join algorithm may be selected by the SQL optimizer when sufficient memory is available to the system, and the BUFFERSIZE option is large enough to store the smaller of the tables into memory.

INFLUENCING THE SQL OPTIMIZER WITH MAGIC

The SQL procedure supports several "magic" options, as is illustrated in the following table, to influence (or force) the use of a specific join algorithm. When a "MAGIC=10x" option is specified, the SQL optimizer's responsibility for choosing the "best" join algorithm for the execution of the join query is essentially suspended and placed in the hands of the user. Although any of these "MAGIC=10x" options can be specified whenever desired, they typically are used when conducting test, evaluation, and performance tuning activities, and avoided in production-level queries.

Option	Description
MAGIC=101	Influences the SQL optimizer to select the Nested Loop join algorithm.
MAGIC=102	Influences the SQL optimizer to select the Sort-Merge join algorithm.
MAGIC=103	Influences the SQL optimizer to select the Hash join algorithm.

In the next example, the tables CUSTOMERS and MOVIES are specified in an equijoin construct, as illustrated below. Each table has a common column, CUST_NO which connects rows together when the value of CUST_NO is equal, and is a way to restrict what rows will be included in the join results.



Specifying MAGIC=101

The following SQL procedure code and corresponding SAS® Log shows the MAGIC=101 option as the option of choice to influence the optimizer in selecting a nested loop join algorithm for executing the join query.

SQL Code

```
PROC SQL MAGIC=101;
  SELECT  *  FROM CUSTOMERS, MOVIES
      WHERE CUSTOMERS.CUST_NO  =  MOVIES.CUST_NO;
QUIT;
```

Log Results

```
PROC SQL MAGIC=101;
  SELECT  *  FROM CUSTOMERS, MOVIES
      WHERE CUSTOMERS.CUST_NO  =  MOVIES.CUST_NO;
NOTE: PROC SQL planner chooses sequential loop join.
QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

Specifying MAGIC=102

The following SQL procedure code and corresponding SAS Log shows the MAGIC=102 option being specified to influence the optimizer in selecting a sort-merge join algorithm for executing the join query.

SQL Code

```
PROC SQL MAGIC=102;
  SELECT  *  FROM CUSTOMERS, MOVIES
      WHERE CUSTOMERS.CUST_NO  =  MOVIES.CUST_NO;
QUIT;
```

Log Results

```
PROC SQL MAGIC=102;
  SELECT  *  FROM CUSTOMERS, MOVIES
      WHERE CUSTOMERS.CUST_NO  =  MOVIES.CUST_NO;
NOTE: PROC SQL planner chooses merge join.
QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.15 seconds
      cpu time            0.04 seconds
```

Specifying MAGIC=103

The following SQL procedure code and corresponding SAS Log shows the MAGIC=103 option being specified to influence the optimizer in selecting a hash join algorithm for executing the join query.

SQL Code

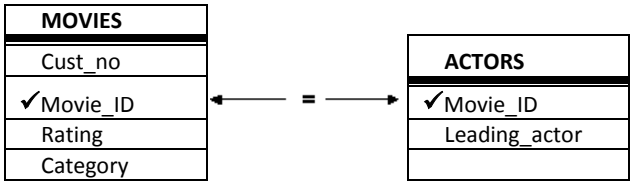
```
PROC SQL MAGIC=103;
  SELECT  *  FROM CUSTOMERS, MOVIES
      WHERE CUSTOMERS.CUST_NO  =  MOVIES.CUST_NO;
QUIT;
```

Log Results

```
PROC SQL MAGIC=103;
  SELECT  *  FROM CUSTOMERS, MOVIES
    WHERE CUSTOMERS.CUST_NO  =  MOVIES.CUST_NO;
NOTE: PROC SQL planner chooses merge join.
NOTE: A merge join has been transformed to a hash join.
QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.15 seconds
      cpu time            0.04 seconds
```

Specifying the IDXWHERE= Data Set Option

In the next example, the tables MOVIES and ACTORS are specified in an equijoin construct, as illustrated below. Each table has a common column, MOVIE_ID which connects rows together when the value of MOVIE_ID is equal, and, as mentioned earlier, is a way to restrict what rows will be included in the join results.



The IDXWHERE= data set option can be specified to influence the SQL optimizer to use the most efficient available index (if one exists) to execute a query. Rather than processing the rows in one or more tables sequentially, specifying an IDXWHERE= data set option, forces the optimizer to identify the most efficient index to use during processing. Care should be used to avoid using this data set option with a small table since it can impede performance because the SAS software would need to traverse the index looking for matches instead of allowing the software to process data using a sequential table scan.

SQL Code

```
PROC SQL;
  SELECT MOVIES.MOVIE_ID, RATING, LEADING_ACTOR
    FROM MOVIES(IDXWHERE=Yes),  ACTORS
      WHERE MOVIES.MOVIE_ID =  ACTORS.MOVIE_ID;
QUIT;
```

Specifying the BUFFERSIZE= Option

If you have surplus virtual memory, you can achieve faster access to matching rows from one or more small input tables by using **Hash** techniques. The **BUFFERSIZE=** option can be used to let the SQL procedure take advantage of hash techniques on larger join tables. The default BUFFERSIZE=n option is 64000 when not specified. In the next example, a BUFFERSIZE=256000 is specified to utilize available memory to load rows. The result is faster performance because additional memory is available to conduct the join reducing the number of data swaps the SAS System has to perform from the slower secondary storage.

SQL Code

```
PROC SQL _method BUFFERSIZE=256000;
  SELECT MOVIES.MOVIE_ID, RATING, LEADING_ACTOR
    FROM MOVIES,  ACTORS
      WHERE MOVIES.MOVIE_ID  =  ACTORS.MOVIE_ID;
QUIT;
```

Log Results

```
NOTE: SQL execution methods chosen are:
      sqxslct
          sqxjhsh
              sqxsrc( MOVIES )
              sqxsrc( ACTORS )
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.04 seconds
      cpu time            0.03 seconds
```

CONCLUSION

The SQL procedure is a powerful and flexible database language for performing a multitude of operations, including joining tables of data. This paper highlighted a variety of procedure options to influence (or force) the SQL optimizer to use a join algorithm that it may not ordinarily choose on its own. Several SQL procedure and table options were illustrated: MAGIC=101, MAGIC=102, MAGIC=103, IDXWHERE=Yes, and BUFFERSIZE= along with their uses to force the SQL optimizer to select a specific join algorithm when conducting test, debug, and performance tuning activities.

REFERENCES

Lafler, Kirk Paul (2013). *PROC SQL: Beyond the Basics Using SAS*, 2nd Edition, SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2013), *"Add a Little Magic to Your Joins,"* Michigan SAS® Users Group (MISUG) 2013 One-day Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), *"Add a Little Magic to Your Joins,"* Iowa SAS® Users Group (IASUG) and Nebraska SAS® Users Group (NebSUG) 2013 Conferences, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Add a Little Magic to Your Joins,"* South Central SAS® Users Group (SCSUG) and MidWest SAS® Users Group (MWSUG) Conferences, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Demystifying PROC SQL Join Algorithms,"* Western Users of SAS® Software (WUSS) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Powerful, But Sometimes Hard-to-find PROC SQL Features,"* Iowa SAS® Users Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2011), *"Powerful and Sometimes Hard-to-find PROC SQL Features,"* PharmaSUG 2011 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2010), *"Exploring Powerful Features in PROC SQL,"* SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2008), *"Kirk's Top Ten Best PROC SQL Tips and Techniques,"* Wisconsin Illinois SAS Users Conference (June 26th, 2008), Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2008), *"Exploring the Undocumented PROC SQL _METHOD Option,"* Proceedings of the SAS Global Forum (SGF) 2008 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2007), *"Undocumented and Hard-to-find PROC SQL Features,"* Proceedings of the PharmaSUG 2007 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul and Ben Cochran (2007), *"A Hands-on Tour Inside the World of PROC SQL Features,"* Proceedings of the SAS Global Forum (SGF) 2007 Conference, Software Intelligence Corporation, Spring Valley, CA, and The Bedford Group, USA.

Lafler, Kirk Paul (2006), *"A Hands-on Tour Inside the World of PROC SQL,"* Proceedings of the 31st Annual SAS Users Group International Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2004). *PROC SQL: Beyond the Basics Using SAS*, SAS Institute Inc., Cary, NC, USA.

TRADEMARK CITATIONS

AUTHOR INFORMATION

Kirk Paul Lafler is an entrepreneur, consultant, educator and author, and has used SAS software since 1979. Currently, Kirk works at San Diego State University as a lecturer and adjunct professor; at the University of California San Diego Extension as an advisor and adjunct professor; and teaches SAS, SQL, Python and R courses, seminars, workshops and webinars to users around the world. As the author of PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019), Google® Search Complete (Odyssey Press. 2014) and hundreds of SAS papers and articles; Kirk has served as an Invited speaker, educator, keynote and section leader at SAS user group conferences and meetings worldwide; and is the recipient of 25 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Kirk Paul Lafler
SAS® Consultant, Application Developer, Programmer, Data Analyst, Educator and Author
E-mail: KirkLafler@cs.com
LinkedIn: https://www.linkedin.com/in/KirkPaulLafler/
LinkedIn: https://www.linkedin.com/in/Order-of-Magnitude-Analytics/
Twitter: @sasNerd