SESUG Paper 33-2021

Under the Hood: The Mechanics of SQL Query Optimization Techniques

Kirk Paul Lafler, @sasNerd

Abstract

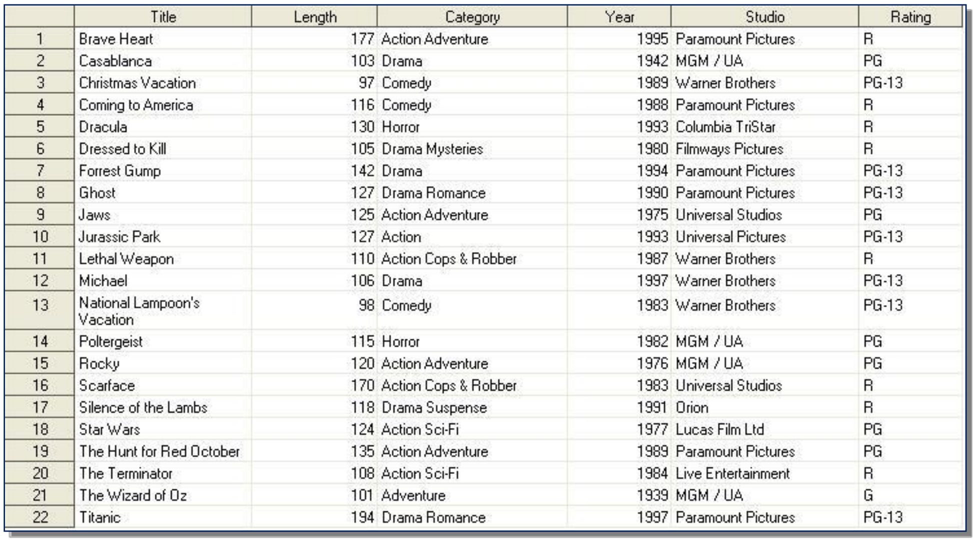
The SAS® software and SQL procedure provide powerful features and options for users to gain a better understanding of what's taking place during query processing. This presentation explores the fully supported SAS® MSGLEVEL=I system option and PROC SQL _METHOD option to display valuable informational messages on the SAS® Log about the SQL optimizer's execution plan as it relates to processing SQL queries; along with an assortment of query optimization techniques.

Introduction

PROC SQL supports a powerful option called **_METHOD**. Since its implementation, many SAS® SQL users have expressed very favorable comments for the value-added information it provides on the SAS Log. In fact, the _METHOD option is worth exploring simply due to the benefits associated with gaining a better understanding of the processes during specific PROC SQL operations, query evaluation, algorithm selected by the optimizer and used in the processing of a query, or testing and debugging operations.

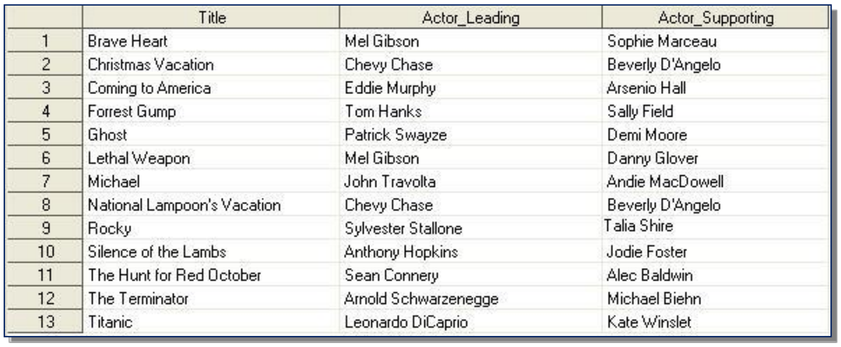
Tables Used in Examples

The data used in all the examples in this paper uses the movies and actors data sets (tables). The Movies table, below, consists of twenty-two observations (rows) and six variables (columns): Title, Length, Category, Year, Studio, and Rating. Title, Category, Studio, and Rating are defined as character columns with Length and Year being defined as numeric columns.



	Title	Length	Category	Year	Studio	Rating
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
2	Casablanca	103	Drama	1942	MGM / UA	PG
3	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
4	Coming to America	116	Comedy	1988	Paramount Pictures	R
5	Dracula	130	Horror	1993	Columbia TriStar	R
6	Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
7	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
8	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
9	Jaws	125	Action Adventure	1975	Universal Studios	PG
10	Jurassic Park	127	Action	1993	Universal Pictures	PG-13
11	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
12	Michael	106	Drama	1997	Warner Brothers	PG-13
13	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
14	Poltergeist	115	Horror	1982	MGM / UA	PG
15	Rocky	120	Action Adventure	1976	MGM / UA	PG
16	Scarface	170	Action Cops & Robber	1983	Universal Studios	R
17	Silence of the Lambs	118	Drama Suspense	1991	Orion	R
18	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
19	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG
20	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
21	The Wizard of Oz	101	Adventure	1939	MGM / UA	G
22	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13

The ACTORS data set (table) consists of thirteen observations (rows) and three variables (columns): Title, Actor_Leading, and Actor_Supporting which are all character columns, and is illustrated below.



	Title	Actor_Leading	Actor_Supporting
1	Brave Heart	Mel Gibson	Sophie Marceau
2	Christmas Vacation	Chevy Chase	Beverly D'Angelo
3	Coming to America	Eddie Murphy	Arsenio Hall
4	Forrest Gump	Tom Hanks	Sally Field
5	Ghost	Patrick Swayze	Demi Moore
6	Lethal Weapon	Mel Gibson	Danny Glover
7	Michael	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	Chevy Chase	Beverly D'Angelo
9	Rocky	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	Sean Connery	Alec Baldwin
12	The Terminator	Arnold Schwarzenegge	Michael Biehn
13	Titanic	Leonardo DiCaprio	Kate Winslet

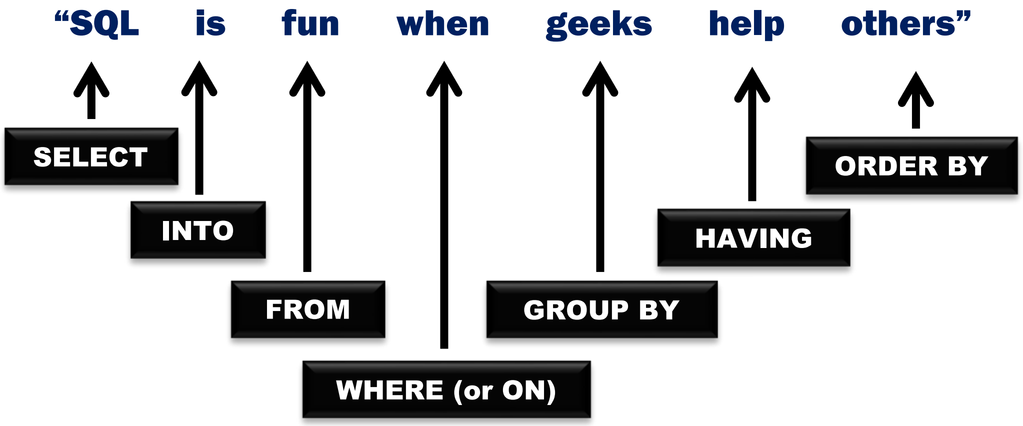
SELECT Clause Syntax Order

A SELECT query retrieves data from one or more database tables. Constructing an SQL SELECT query begins by specifying one or more columns and a required FROM-clause. The remaining SELECT clauses are optional, but, if specified, must adhere to basic syntax rules. The syntax order of the various SELECT clauses is:

```
PROC SQL ;
  SELECT  .   .   .
    INTO   .   .   .
      FROM   .   .   .
        WHERE   < OR >   ON   .   .   .
          GROUP BY   .   .   .
            HAVING   .   .   .
              ORDER BY   .   .   . ;
QUIT ;
```

SELECT Clause Execution Order

The SELECT statement's purpose is to retrieve (or read) data from one, or more, underlying tables (or views). Although the SELECT statement supports multiple clauses, only one clause is required – the FROM clause. All remaining clauses are optional and only specified as needed. To help remember the specific order of the SELECT statement's clauses, recite this phrase:



Execution Order	Description
1. FROM	The first clause executed in a query is the FROM clause. It's a required clause with the purpose of determining the working set of data that is being queried (i.e., variable names, variable type, number of rows, and other important information).
2. INTO	Creates one or more macro variables where the values can be used to manipulate data.
3. ON	Subsets rows of data based on the condition(s) specified, and rows that aren't satisfied by the condition(s) are discarded.
4. WHERE	Subsets rows of data based on the condition(s) specified, and rows that aren't satisfied by the condition(s) are discarded.
5. GROUP BY	Processes the rows that were subset with the WHERE clause and grouped based on common values in the column specified in the GROUP BY clause.
6. HAVING	Applies the condition(s) to the grouped rows specified in the GROUP BY clause, and any grouped rows that aren't satisfied by the condition(s) are discarded.
7. SELECT	Columns and expressions specified in the SELECT statement are processed.
8. ORDER BY	Arranges the rows of data in either ascending (default) or descending order.

Displaying Informational SAS Log Messages with MSGLEVEL=

SAS users can control how much information the SAS System writes to the SAS log by specifying the MSGLEVEL= SAS System option in an Options statement. The MSGLEVEL= option supports two possible values: **N** (which is the default) to print standard notes, warnings, and error messages; and **I** to print standard notes, warnings, error messages, plus additional information about sort, merge, and index processing. When specifying **MSGLEVEL=I** in an options statement, SAS displays the sort product that was used in a sort operation, a warning when variables are overwritten during merge processing; and the name of the available index that was used in index processing (or helpful suggestions on what can be done to influence SAS to use an available index); along with the usual assortment of notes, warnings, and error messages.

To demonstrate the effect of a **MSGLEVEL=I** option statement the following example illustrates a simple SQL join query on two tables, MOVIES and ACTORS. As shown in the resulting SAS Log, an informative message was generated explaining that the SAS system chose to use an available index, Rating, to optimize WHERE clause processing. This use of the MSGLEVEL=I system option provides users with a better understanding of what the SAS system did to improve processing, as well as the specific name of the index that was selected during processing of the query.

SQL Code

```
OPTIONS MSGLEVEL=I ;
PROC SQL ;
  SELECT MOVIES.TITLE, RATING, LENGTH, ACTOR_LEADING
    FROM MOVIES,
         ACTORS
      WHERE MOVIES.TITLE = ACTORS.TITLE AND RATING = 'PG' ;
QUIT ;
```

Log Results

```
OPTIONS MSGLEVEL=I ;
PROC SQL ;
  SELECT MOVIES.TITLE, RATING, LENGTH, ACTOR_LEADING
    FROM MOVIES,
         ACTORS
      WHERE MOVIES.TITLE = ACTORS.TITLE AND RATING = 'PG' ;

INFO: Index Rating selected for WHERE clause optimization.

QUIT ;
```

PROC SQL Join Algorithms

When it comes to performing PROC SQL joins, users supply the names of the tables for joining along with the join conditions, and the PROC SQL optimizer determines which of the four available join algorithms to use for performing the join query operation. The four join algorithms available to the optimizer include:

- ✓ **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.

The _Method Option and Code Descriptions

The PROC SQL _METHOD option can be specified as an effective way to analyze a query process or for debugging purposes. Processing information from the _METHOD option is automatically displayed on the Log using a variety of codes. The complete list of codes available with the _METHOD option along with their corresponding descriptions is displayed in the following table.

Code	Description
SQXCRTA	**Create table as Select.**
SQXSLCT	**Select statement or clause.**
SQXJSL	**Step loop join (Cartesian).**
SQXJM	**Merge join operation.**
SQXJNDX	**Index join operation.**
SQXJHSH	**Hash join operation.**
SQXSORT	**Sort operation.**
SQXSRC	**Source rows from table.**
SQXFIL	**Rows filtration.**
SQXSUMG	**Summary stats (aggregates) with GROUP BY clause.**
SQXSUMN	**Summary stats with no GROUP BY clause.**

Application of the _METHOD Option

In the following example a _METHOD option is specified to show the processing hierarchy in a two-way equi-join. As illustrated in the SAS Log, the PROC SQL optimizer utilized a hash join algorithm in the performance of the join query.

<u>SQL Code</u>

```
OPTIONS MSGLEVEL=I ;
PROC SQL _METHOD ;
   SELECT MOVIES.TITLE, RATING, ACTOR_LEADING
      FROM MOVIES,
           ACTORS
         WHERE MOVIES.TITLE = ACTORS.TITLE AND RATING = 'PG' ;
QUIT ;
```

<u>Log Results</u>

```
OPTIONS MSGLEVEL=I ;
PROC SQL _METHOD ;
  SELECT MOVIES.TITLE, RATING, LENGTH, ACTOR_LEADING
    FROM MOVIES,
         ACTORS
      WHERE MOVIES.TITLE = ACTORS.TITLE AND RATING = 'PG' ;
NOTE: SQL execution methods chosen are:
      sqxslct
          sqxjhsh
              sqxsrc( MOVIES )
              sqxsrc( ACTORS )
INFO: Index Rating selected for WHERE clause optimization.
QUIT ;
```

SQL Query Optimization Techniques

A number of query optimization techniques are available to SAS and SQL users. We'll explore a few of these optimization techniques including STIMER / FULLSTIMER system options, SEELECT clause execution order, logic construction, and the construction and application of indexes.

STIMER and FULLSTIMER System Options

When turned on, the STIMER and FULLSTIMER system options provide SAS and SQL users with measurable resource results associated with CPU, I/O, memory, and data storage usage. The results provide users with a way to compare and contrast the various techniques, and to explore and evaluate performance results to assist in achieving an optimal balance between competing computer resources. The level of detail produced as a result of turning STIMER and FULLSTIMER system options on can be seen in the following SAS Log results.

- **Options STIMER;**

```
NOTE: DATA statement used (Total process time):
      real time          0.04 seconds
      cpu time           0.03 seconds
```

- **Options FULLSTIMER;**

```
NOTE: DATA statement used (Total process time):
      real time                 0.05 seconds
      user cpu time             0.03 seconds
      system cpu time           0.04 seconds
      memory                    661.23k
      OS Memory                 6768.00k
      Timestamp                 09/16/2012 11:17:19 PM
```

Logic Construction

Logic conditions affect processing costs. The SQL optimizer evaluates a series of "ANDed" expressions in a WHERE (or ON) from left to right. A chain of "ANDed" conditions should be specified with the most restrictive expression first. As a result, fewer resources are spent by bypassing rows that do not satisfy the first conditional value in the WHERE (or ON) clause.

The Construction and Application of Indexes

Given the number of books and articles on SQL and SQL-related topics, I find it strange that there is not discussion related to indexes and their impact on WHERE (and ON) clause processing. Certainly, these topics deserve additional attention in order to assist SQL users' improver their understanding in applying these powerful features in their application of database queries.

Understanding Indexes

What exactly is an index? An index consists of one or more columns in a table to uniquely identify each row of data within the table. Operating as a SAS object containing the values in one or more columns in a table, an index is comprised of one or more columns and may be defined as numeric, character, or a combination of both. Although there is no rule that says a table must have an index, when present, they are most frequently used to make information retrieval using a WHERE (or ON) clause more efficient.

Indexes are designed to improve the speed in which subsets of data are accessed. Rather than physically sorting a table (as performed by the ORDER BY clause or the BY statement in PROC SORT), an index is designed to set up a logical data arrangement without the need to physically sort it. This has the advantage of reducing CPU and memory requirements. It also reduces data access time when using WHERE clause processing.

To help determine when an index is necessary, it is important to look at existing data as well as the way the base table(s) will be used. It is also critical to know what queries will be used and how they will access columns of data. There are times when the column(s) making up an index are obvious and other times when they are not. When determining whether an index provides any processing value, some very important rules should be kept in mind. An index should permit the greatest flexibility so every column in a table can be accessed and displayed. Indexes should also be assigned to discriminating column(s) only since query results will benefit greatest when this is the case.

Simple Rules to Know About Indexes

When an index is specified on one or more tables, a join process may actually be boosted. The PROC SQL processor may use an index, when certain conditions permit its use. Here are a few things to keep in mind when creating an index:

- If the table is small, sequential processing may be just as fast, or faster, than processing with an index;
- Avoid creating more indexes than are absolutely necessary;
- If the page count, as displayed in the CONTENTS procedure, is less than 3 pages, an index may provide little or no value;
- If the data subset for the index is large, sequential access may be more efficient than using the index;
- If the percentage of matches is 15% or less (known as the 15% rule) of the overall population then an index may be beneficial;
- The costs associated with maintaining an index can outweigh its performance value, because an index is updated each time the rows in a table are added, deleted, or modified.

Sample code is illustrated next to demonstrate the creation of simple and composite indexes using the CREATE INDEX statement in the SQL procedure.

Creating a Simple Index

A simple index is specifically defined for one column in a table and must be the same name as the column. Suppose you had to create an index consisting of movie rating (RATING) in the MOVIES table. Once created, the index becomes a separate object located in the SAS library.

SQL Code

```
   PROC SQL;
     CREATE INDEX RATING ON MOVIES(RATING);
   QUIT;
```

SAS Log Results

```
   PROC SQL;
     CREATE INDEX RATING ON MOVIES(RATING);
NOTE: Simple index RATING has been defined.
   QUIT;
```

Creating a Composite Index

A composite index is defined for two or more columns in a table and must have a **unique** name that is different than the columns assigned to the index. Suppose you were to create an index consisting of movie category (CATEGORY) and movie rating (RATING) in the MOVIES table. Once the composite index is created, the index consisting of the two table columns become a separate object located in the SAS library.

SQL Code

```
   PROC SQL;
     CREATE INDEX CAT_RATING ON MOVIES(CATEGORY, RATING);
   QUIT;
```

SAS Log Results

```
   PROC SQL;
     CREATE INDEX CAT_RATING ON MOVIES(CATEGORY, RATING);
NOTE: Composite index CAT_RATING has been defined.
   QUIT;
```

Index Entries and Pointers

An index file is stored in the same SAS library as its associated data file. Having the same name as its data file, it is represented as a separate entity known as an INDEX member type. An index file contains entries organized hierarchically with entries being connected by pointers and organized in ascending order. Each entry contains a unique value corresponding to the column's frequency distribution and one or more unique observations, referred to as the record identifier (RID). Space that is occupied by deleted values are automatically reclaimed and reused by the index. A sample index containing entries representing the index file for the movie rating (RATING) is illustrated below.

Value	Record ID (RID)
G	21
PG	2, 9, 14, 15, 18, 19
PG-13	3, 7, 8, 10, 12, 13, 22
R	1, 4, 5, 6, 11, 16, 17, 20

Index Limitations

Indexes can be very useful, but they do have limitations. As data in a table is inserted, modified, or deleted, an index must also be updated by the SAS System to address any and all changes. This automatic feature requires additional CPU resources to process changes to a table. Also, as a separate structure in its own right, an index can consume considerable storage space. As a consequence, care should be exercised not to create too many indexes but assign indexes to the most discriminating variables in a table.

Because of the aforementioned drawbacks, indexes should only be created on tables where query search time needs to be optimized. Any unnecessary indexes may force the SAS System to expend unnecessary resources updating and reorganizing after insert, delete, and update operations are performed. Also, select one or more columns to represent an index that has a subset size of no more than 15% (or smaller) of the population data set. This is sometimes referred to as the 15% rule.

Optimizing Where (or ON) Clause Processing With Indexes

A WHERE clause defines the logical conditions that control which rows a SELECT statement will select, a DELETE statement will delete, or an UPDATE statement will update. This powerful, but optional, clause permits SAS users to test and evaluate conditions as true or false. From a programming perspective, the evaluation of a condition determines which of the alternate paths a program will follow. Conditional logic in PROC SQL is frequently implemented in a WHERE clause to reference constants and relationships between columns and data values.

To achieve the best possible performance from programs containing SQL procedure code, the SQL optimizer determines whether any available index(es) will perform better than if it were to use more traditional sequential data access. Many users incorrectly assume that an available index is automatically used with WHERE-clause processing, but this is not always the case. In fact, WHERE-clause processing does nothing more than influence the SQL optimizer to take advantage of an index. When the optimizer determines that an index will improve processing speeds, the index is used to direct activities related to data access. Otherwise, the SQL optimizer uses the more traditional, and default, sequential data access method with WHERE-clause processing.

Conclusion

SAS users are encouraged to learn and apply various optimization techniques when using SQL. From the MSGLEVEL=I SAS System option, the _METHOD PROC SQL option, the STIMER and FULLSTIMER system options, understanding the SELECT clause execution order, logic construction, and the application of indexes, users should become familiar with the application of these incredible optimization techniques. When used, SAS and SQL users have effective tools to display useful information and achieve greater insight into the processes performed during specific PROC SQL operations, including query evaluation, the algorithm that is selected and used by the SQL optimizer in the processing of a query's index, testing and debugging, and other application processes.

References

Lafler, Kirk Paul (2019). *PROC SQL: Beyond the Basics Using SAS, Third Edition*, SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2016), *"Triggering the SAS SQL Execution Plan,"* Gateway Area of Users of SAS Software (GAUSS) 2016 Meeting, Software Intelligence Corporation, Spring Valley, CA, USA.

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

Lafler, Kirk Paul and Charlie Shipp (2013), *"Exploring the PROC SQL _METHOD Option,"* Proceedings of the 2013 Western Users of SAS Software (WUSS) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul and Charlie Shipp (2013), *"Exploring the PROC SQL _METHOD Option,"* Proceedings of the 2013 MidWest SAS Users Group (MWSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul and Charlie Shipp (2013), *"Exploring the PROC SQL _METHOD Option,"* Proceedings of the 2013 NorthEast SAS Users Group (NESUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2013), *"Exploring the PROC SQL _METHOD Option,"* Proceedings of the 2013 SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2012), *"Exploring the PROC SQL _METHOD Option,"* Proceedings of the 2012 MidWest SAS Users Group (MWSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2010), *"DATA Step and PROC SQL Programming Techniques,"* Ohio SAS Users Group (OSUG) 2010 One-Day Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009), *"Exploring DICTIONARY Tables and SASHELP Views,"* Proceedings of the 2009 South Central SAS Users Group (SCSUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009), *"Exploring DICTIONARY Tables and SASHELP Views,"* Proceedings of the 2009 Western Users of SAS Software (WUSS) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009), *"Exploring DICTIONARY Tables and SASHELP Views,"* Proceedings of the 2009 PharmaSUG SAS Users Group 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 2008 Western Users of SAS Software (WUSS) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2007), *"Undocumented and Hard-to-Find PROC SQL Features,"* Proceedings of the 2007 NorthEast SAS Users Group (NESUG) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

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

Lafler, Kirk Paul (2003), "*Undocumented and Hard-to-find PROC SQL Features*," Proceedings of the 2007 Western Users of SAS Software (WUSS) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2002). *PROC SQL Programming Tips*; Software Intelligence Corporation, Spring Valley, CA, USA.

Shipp, Charles Edwin and Kirk Paul Lafler (2013), *"Exploring the PROC SQL _METHOD Option,"* Proceedings of the 2013 SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Acknowledgments

The author thanks the 2021 SESUG Conference Committee, particularly Know You SAS: Advanced Section Chairs Nat Wooding, Barbara Okerson and Andrea Lewton for accepting my abstract and paper; the 2021 SESUG Executive Committee for organizing and supporting this virtual conference event; SAS Institute Inc. for providing SAS users with wonderful software; and SAS users everywhere for being the nicest people anywhere!

Trademark Citations

Author Information

Kirk Paul Lafler is a lecturer and adjunct professor at San Diego State University; an advisor and adjunct professor at the University of California San Diego Extension; and teaches SAS, SQL, Python, R and Excel courses, seminars, workshops, and webinars to students, professionals and users around the world. Kirk has been a SAS user since 1979 and is the author of several books including, PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019) along with papers and articles on a variety of SAS topics. Kirk has also been selected as an Invited speaker, educator, keynote and section leader at SAS conferences; 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® / SQL 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