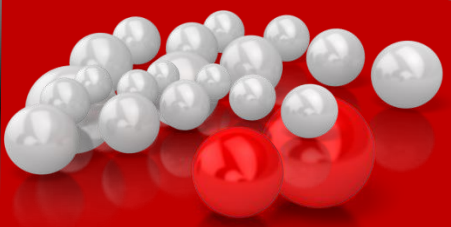
Data-driven Programming Techniques Using SAS®

Presenter Biography

Kirk Paul Lafler is an entrepreneur, consultant and programmer, and has used SAS software since 1979. Kirk currently works as a SAS consultant, application developer, programmer, data analyst and educator; a lecturer and adjunct professor at San Diego State University; an advisor and adjunct professor at the University of California San Diego Extension; and an educator of dozens of SAS, SQL, R and Python courses, seminars, workshops, and webinars to thousands of users around the world.

As the author of several books including PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019) along with hundreds of papers and articles on a variety of SAS topics; Kirk has been selected as an Invited speaker, educator, keynote and section leader at SAS conferences and meetings worldwide; and is the recipient of 25 “Best” contributed paper, hands-on workshop (HOW), and poster awards.



Presentation Topics

Popular Programming Paradigms

Data-driven Programming Techniques Using SAS Metadata Sources

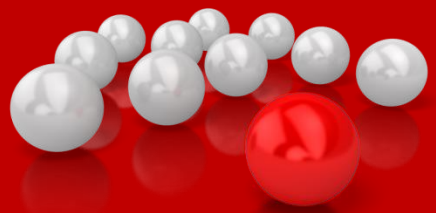
Data-driven Programming Techniques Using CALL EXECUTE, User-defined Formats, and PROC SQL / Macros

Table Examples – SASHELP.CARS

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	Engine Size	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	179
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	180
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3.0	6	220	20	28	3462	104	179
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3.0	6	220	17	26	3583	104	179
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3.0	6	220	18	25	3627	104	179
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3.0	6	220	20	27	3561	109	192
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3.0	6	220	18	25	3880	109	192
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	$42,490	$38,325	3.0	6	220	20	27	3814	105	180
16	Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	$44,240	$40,075	3.0	6	220	18	25	4013	105	180
17	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	$42,840	$38,840	2.7	6	250	18	25	3836	109	192
18	Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	$49,690	$44,936	4.2	8	300	17	24	4024	109	193
19	Audi	A8 L Quattro 4dr	Sedan	Europe	All	$69,190	$64,740	4.2	8	330	17	24	4399	121	204
20	Audi	S4 Quattro 4dr	Sedan	Europe	All	$48,040	$43,556	4.2	8	340	14	20	3825	104	179
21	Audi	RS 6 4dr	Sports	Europe	Front	$84,600	$76,417	4.2	8	450	15	22	4024	109	191

428 Observations
15 Variables

Popular Programming Paradigms



Programming Paradigms

Programming languages are often classified by their basic features into one of the programming paradigms. Three popular programming paradigms are in use today:

- ✓ **Procedural programming** – represented by blocks of code being organized logically by function, such as data input, data processing or manipulation, and data / results;
- ✓ **Object-oriented programming** – represented by a combination of functionality (behaviors) and data (attributes) hidden inside an object which can then be arranged into classes;
- ✓ **Data-driven programming** – represented by data controlling the flow of execution in a program.

What is Data-driven Programming?

Unlike procedural programming languages and object-oriented programming, data-driven programming involves decisions and processes that are controlled by data.

Why Design Data-driven Programs?

- Data-driven programs can adapt to different data sets and structures;
- Data is used to define the control flow of a program;
- Data-driven programs can write other programs;
- Avoid “hard-coding” approaches which are not only harder to maintain, but increase the cost to support the code over its operational life;
- Reduce programming and maintenance efforts;
- Data-driven programs can be designed for reusability which translates into cost savings.

Data-driven Programming Approaches

Four data-driven programming approaches will be presented:

- ✓ **Using SAS Metadata Dictionary tables and SASHELP views;**
- ✓ **Using the CALL EXECUTE routine;**
- ✓ **Using User-defined informats and formats;**
- ✓ **Using PROC SQL SELECT and the Macro language.**

Order of PROC SQL SELECT Clauses

The PROC SQL 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 used as needed.

Order of PROC SQL SELECT Clauses

To help remember the specific order of the SELECT statement's clauses, recite:

"SQL is fun when geeks help others"

The first letter in each word corresponds to the SELECT statement's clause.

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

SELECT Clause Execution Order

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	The INTO clause is used to create one or more macro variables where the values can be used to manipulate data.
3. ON	The ON clause is used to subset rows of data based on the condition(s) specified, and rows that aren't satisfied by the condition(s) are discarded.
4. WHERE	The WHERE clause is used to subset rows of data based on the condition(s) specified, and rows that aren't satisfied by the condition(s) are discarded.
5. GROUP BY	The GROUP BY clause takes 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	The HAVING clause 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	Expressions specified in the SELECT statement are processed.
8. ORDER BY	The ORDER BY clause sorts the rows of data in either ascending (default) or descending order.

Number of Variable Levels

- Lafler (2017) offers a variety of approaches to determine the number of variable levels in any SAS data set;
- Eliminates the need to construct counting routines in programs;
- Uses PROC SQL or PROC FREQ to produce results.

Number of Variable Levels – PROC FREQ

Code:

```
title "Display NLevels for Origin with PROC FREQ" ;
proc freq data=SASHELP.CARS NLEVELS ;
  tables Origin / nopct nocum ;
run ;
```

Results:

Display NLevels for Origin with PROC FREQ

The FREQ Procedure

Number of Variable Levels	
Variable	Levels
Origin	3

Origin	Frequency
Asia	158
Europe	123
USA	147

Number of Variable Levels – PROC SQL

Code:

```
proc sql ;
 title "Display NLevels for Origin with PROC SQL" ;
 select COUNT(DISTINCT Origin) "Unique Levels"
  from SASHELP.CARS ;
 title "Display NLevels and Counts for Origin with PROC SQL";
 select Origin, COUNT(Origin) AS CTR_Origin
  from SASHELP.CARS
   group by Origin ;
quit ;
```

Results:

Display NLevels for Origin with PROC SQL

Unique Levels
3

Display NLevels and Counts for Origin with PROC SQL

Origin	CTR_Origin
Asia	158
Europe	123
USA	147

Data-driven Programming Techniques Using SAS Metadata Sources

What is Metadata?

- Metadata is everywhere;
- Metadata is often referred to as data about data;
- Others define metadata as information about the design and specification of data structures;
- Metadata does not represent the physical data resource, but the information that describes the resource;
- Metadata provides the details (i.e., what, why, when, and where) about the various resources in a structured way.

Why Use Metadata?

Metadata provides easier, faster and more reliable ways to find or locate information about a digital or data asset. The reasons for using metadata are:

- ✓ **Productivity often improves as search access speeds increase;**
- ✓ **User acceptance levels often improve as the reliability and relevancy of results improve;**
- ✓ **Enhanced filtering capabilities by using meaningful metadata variables to provide greater flexibility;**
- ✓ **Improved organization of data resources for searching, sorting and summarizing;**
- ✓ **Automating a process for code and data reusability, code and data sharing, and improved productivity by programmers.**

Traditional Metadata Methods

Metadata has traditionally been produced using:

- ✓ **PROC CONTENTS – Produces a directory of the SAS library and the details associated with each member type stored in a SAS library.**
- ✓ **PROC DATASETS – Raithel (2016) describes PROC DATASETS as the Swiss Army Knife of Data Management procedures. Like PROC CONTENTS, the PROC DATASETS CONTENTS statement produces a directory of the SAS library and the details associated with each member type (e.g., DATA, VIEW, INDEX) stored in a SAS library.**

Dictionary Tables and SASHELP Views

- SAS collects information about a session;
- The session information is captured as read-only content;
- Dictionary tables are accessible using PROC SQL:
  - ✓ Specify table in FROM clause of a SELECT
  - ✓ DICTIONARY libref is automatically assigned

```
PROC SQL ;
 SELECT *
  FROM DICTIONARY.table-name ;
QUIT ;
```

- SASHELP views can be accessed in a DATA step or with any of your favorite PROCs.

Viewing Dictionary Tables/SASHELP Views

- **# of DICTIONARY Tables and SASHELP Views:**
  - ✓ **22 in SAS 9.1.x;**
  - ✓ **29 in SAS 9.2;**
  - ✓ **30 in SAS 9.3;**
  - ✓ **32 in SAS 9.4 (39 Views in SAS 9.4).**

A Sampling of Dictionary Tables

Table Name	Description
CATALOGS	Allocated SAS catalog librefs and member names.
CHECK_CONSTRAINTS	User-assigned CHECK constraints.
COLUMNS	Column names and attribute information.
DICTIONARIES	Information about all known Dictionary tables.
EXTFILES	FILEREF, pathname, and engine for external files.
FORMATS	Information about defined formats and informats.
INDEXES	Data set index information for each libref.
LIBNAMES	Allocated librefs and paths.
MACROS	Macro variables, their scope, and value.
OPTIONS	SAS system options and settings of your environment.
STYLES	Librefs, template names, style names, and notes.
TABLES	Allocated SAS data set librefs and member names.
VIEWS	Librefs and dictionary view names.

A Sampling of SASHELP Views

View Name	Description
VCATLG	Allocated SAS catalog librefs and member names.
VCHKCON	User- and system-assigned CHECK constraints.
VCOLUMN	Column names and attribute information.
VDCTNRY	Information about all known Dictionary tables.
VEXTFL	FILEREF, pathname, and engine for external files.
VFORMAT	Information about defined formats and informats.
VINDEX	Data set index information for each libref.
VLIBNAM	Allocated librefs and paths.
VMACRO	Macro variables, their scope, and value.
VOPTION	SAS system options and settings of your environment.
VSTYLE	Librefs, template names, style names, and notes.
VTABLE	Allocated SAS data set librefs and member names.
VVIEW	Librefs and dictionary view names.

Dictionary.TABLES or SASHELP.VTABLE

- Dictionary.TABLES Table
- SASHELP.VTABLE View
- 41 Variables (or Columns)

Library Name	Member Name	Member Type	DBMS Member Type	Data Set Label	Data Set Type	Date Created	Date Modified	Number of Physical Observations	Observation Length
SASHELP	CARS	DATA		2004 Car Data	DATA	07NOV18:20:48:08	07NOV18:20:48:08	428	152

Number of Variables	Type of Password Protection	Compression Routine	Encryption	Number of Pages	Size of File	Percent Compression	Reuse Space	Bufsize	Number of Deleted Observations	Number of Logical Observations	Longest variable name	Longest label
15	---	NO	NO	2	196608	0	no	65536	0	428	11	15

Maximum number of generations	Generation number	Data Set Attributes	Type of Indexes	Data Representation	Name of Collating Sequence	Sorting Type	Charset Sorted By	Requirements Vector	Data Representation Name
0	.	ON		NATIVE		S	ANSI	181F101122220033330102320433012 3	WINDOWS_64

Data Encoding	Audit Trail Active?	Audit Before Image?	Audit Admin Image?	Audit Error Image?	Audit Data Image?	Number of Character Variables	Number of Numeric Variables
us-ascii ASCII (ANSI)	no	no	no	no	no	5	10

Dictionary.COLUMNS or SASHELP.VCOLUMN

- Dictionary.COLUMNS Table
- SASHELP.VCOLUMN View
- 18 Variables (or Columns)

Library Name	Member Name	Member Type	Column Name	Column Type	Column Length	Column Position	Column Number in Table	Column Label	Column Format	Column Informat	Column Index Type	Order in Key Sequence	Extende Type	Not NULL?	Precision	Scale	Transcoded
SASHELP	CARS	DATA	Make	char	13	80	1					1	char	no	0	.	yes
SASHELP	CARS	DATA	Model	char	40	93	2					0	char	no	0	.	yes
SASHELP	CARS	DATA	Type	char	8	133	3					2	char	no	0	.	yes
SASHELP	CARS	DATA	Origin	char	6	141	4					0	char	no	0	.	yes
SASHELP	CARS	DATA	DriveTrain	char	5	147	5					0	char	no	0	.	yes
SASHELP	CARS	DATA	MSRP	num	8	0	6		DOLLAR8.			0	num	no	0	.	yes
SASHELP	CARS	DATA	Invoice	num	8	8	7		DOLLAR8.			0	num	no	0	.	yes
SASHELP	CARS	DATA	EngineSize	num	8	16	8	Engine Size (L)				0	num	no	0	.	yes
SASHELP	CARS	DATA	Cylinders	num	8	24	9					0	num	no	0	.	yes
SASHELP	CARS	DATA	Horsepower	num	8	32	10					0	num	no	0	.	yes
SASHELP	CARS	DATA	MPG_City	num	8	40	11	MPG (City)				0	num	no	0	.	yes
SASHELP	CARS	DATA	MPG_Highway	num	8	48	12	MPG (Highway)				0	num	no	0	.	yes
SASHELP	CARS	DATA	Weight	num	8	56	13	Weight (LBS)				0	num	no	0	.	yes
SASHELP	CARS	DATA	Wheelbase	num	8	64	14	Wheelbase (IN)				0	num	no	0	.	yes
SASHELP	CARS	DATA	Length	num	8	72	15	Length (IN)				0	num	no	0	.	yes
SASHELP	CLASS	DATA	Name	char	8	24	1					0	char	no	0	.	yes
SASHELP	CLASS	DATA	Sex	char	1	32	2					0	char	no	0	.	yes
SASHELP	CLASS	DATA	Age	num	8	0	3					0	num	no	0	.	yes
SASHELP	CLASS	DATA	Height	num	8	8	4					0	num	no	0	.	yes
SASHELP	CLASS	DATA	Weight	num	8	16	5					0	num	no	0	.	yes

Dictionary.OPTIONS or SASHELP.VOPTION

- Dictionary.OPTIONS Table
- SASHELP.VOPTION View
- 8 Variables (or Columns)

Option Name	Option type	Offset into option value	Option Setting	Option Description	Option Location	Option Set	Option Group
ASYNCHIO	Boolean	0	NOASYNCHIO	Enables asynchronous input and output.	Portable	startup	SASFILES
CATCACHE	num	0	0	Specifies the number of SAS catalogs to keep open in cache memory.	Portable	startup	SASFILES
CBUFNO	num	0	0	Specifies the number of extra page buffers to allocate for each open SAS catalog.	Portable	anytime	SASFILES
CMPLIB	char	0		Specifies one or more SAS data sets that contain compiler subroutines to include during compilation.	Portable	anytime	SASFILES
DATASTMTCHK	char	0	COREKEYWORDS	Specifies which SAS statement keywords are prohibited from being specified as a one-level DATA step name to protect against overwriting an input data set.	Portable	anytime	SASFILES
DKRICOND	char	0	ERROR	Specifies the error level to report when a variable is missing from an input data set during the processing of a DROP=, KEEP=, or RENAME= data set option.	Portable	anytime	SASFILES
DKROCOND	char	0	WARN	Specifies the error level to report when a variable is missing from an output data set during the processing of a DROP=, KEEP=, or RENAME= data set option.	Portable	anytime	SASFILES
DLCREATEDIR	Boolean	0	NODLCREATEDIR	Creates a directory for the SAS library that is named in a LIBNAME statement if the directory does not already exist.	Portable	anytime	SASFILES
DLDMGACTION	char	0	REPAIR	Specifies the type of action to take when a SAS data set or a SAS catalog is detected as damaged.	Portable	anytime	SASFILES
ENGINE	char	0	V9	Specifies the default access method for SAS libraries.	Portable	startup	SASFILES
EXTENDEDDATATYPES	char	0	NO	Specifies whether SAS processes all supported data types or converts nontraditional SAS data types to CHAR and DOUBLE.	Portable	anytime	SASFILES
EXTENDOBSCOUNTER	char	0	YES	Specifies whether to extend the maximum number of observations in a new SAS data file.	Portable	anytime	SASFILES
FILESYNC	char	0	HOST	Specifies when operating system buffers that contain contents of permanent SAS files are written to disk.	Portable	startup	SASFILES
FIRSTOBS	num	0	1	Specifies the observation number or external file record that SAS processes first.	Portable	anytime	SASFILES
IBUFNO	num	0	0	Specifies the number of extra buffers to be allocated for navigating an index file.	Portable	anytime	SASFILES
IBUFSIZE	num	0	0	Specifies the buffer page size for an index file.	Portable	anytime	SASFILES

Rows in All Tables – PROC SQL

Code:

```
PROC SQL ;
 SELECT LIBNAME, MEMNAME, NOBS
  FROM DICTIONARY.TABLES
   WHERE LIBNAME = "SASHELP" ;
QUIT ;
```

Results:

Library Name	Member Name	Number of Physical Observations
SASHELP	AACOMP	2020
SASHELP	AARFM	130
SASHELP	ADSMSG	426
SASHELP	AFMSG	1090
SASHELP	AIR	144
SASHELP	APPLIANC	156
SASHELP	ASSCMGR	402
SASHELP	BASEBALL	322
SASHELP	BEI	24205
SASHELP	BIRTHWGT	100000
SASHELP	BMIMEN	3264
SASHELP	BMT	137
SASHELP	BURROWS	24591
SASHELP	BUY	11
SASHELP	BWEIGHT	50000
SASHELP	CARS	428

Rows in All Tables – PROC PRINT

Code:

```
PROC PRINT DATA=SASHELP.VTABLE NOOBS ;
 VAR LIBNAME MEMNAME NOBS ;
 WHERE LIBNAME = "SASHELP" ;
RUN ;
```

Results:

Library Name	Member Name	Number of Physical Observations
SASHELP	AACOMP	2020
SASHELP	AARFM	130
SASHELP	ADSMSG	426
SASHELP	AFMSG	1090
SASHELP	AIR	144
SASHELP	APPLIANC	156
SASHELP	ASSCMGR	402
SASHELP	BASEBALL	322
SASHELP	BEI	24205
SASHELP	BIRTHWGT	100000
SASHELP	BMIMEN	3264
SASHELP	BMT	137
SASHELP	BURROWS	24591
SASHELP	BUY	11
SASHELP	BWEIGHT	50000
SASHELP	CARS	428

Number of Character/Numeric Variables

- Lafler (2013) offers a way to determine the number of character and numeric variables for any SAS data set;
- Reduce the time to review pages of PROC CONTENTS output;
- Use the DATA step or your favorite PROC;
- Access the contents of the Dictionary.TABLES and SASHELP.VTABLE view;
- Select columns: LIBNAME, MEMNAME, MEMTYPE, NUM_CHARACTER, NUM_NUMERIC, and NVAR.

Number of Character/Numeric Variables

Code:

```
TITLE # Character vs Numeric Variables All Datasets ;
PROC SQL ;
 SELECT LIBNAME,MEMNAME,NUM_CHARACTER,NUM_NUMERIC,NVAR
  FROM DICTIONARY.TABLES
   WHERE MEMTYPE = "DATA" ;
QUIT ;
```

Results:

Character vs Numeric Variables All Datasets

Library Name	Member Name	Number of Character Variables	Number of Numeric Variables	Number of Variables
SASHELP	AACOMP	3	1	4
SASHELP	AARFM	3	1	4
SASHELP	ADSMSG	4	2	6
SASHELP	AFMSG	4	2	6
SASHELP	AIR	0	2	2
SASHELP	APPLIANC	0	25	25
SASHELP	ASSCMGR	12	7	19
SASHELP	BASEBALL	6	18	24
SASHELP	BEI	0	24	24
SASHELP	BIRTHWGT	7	1	8
SASHELP	BMIMEN	0	2	2
SASHELP	BMT	1	2	3
SASHELP	BURROWS	0	14	14
SASHELP	BUY	0	2	2
SASHELP	BWEIGHT	0	10	10
SASHELP	CARS	5	10	15

Variable Keeper – PROC SQL

- Lafler (2013) describes a macro that creates a list of variables for processing in a SELECT query;
- Triggered by calling a macro to reduce coding requirements;
- Assigns the directory and SAS table name that the user specified when calling macro;
- Reads the Dictionary.COLUMNS table and creates a value-list macro variable containing the names of all variables;
- Produces a subset based on what the user enters when calling the macro.

Variable Keeper – PROC SQL

Code:

```
proc sql noprint ;
 select name,
        count(name)
  into :mvariables separated by ', ',
       :mvariablesnum
   from dictionary.columns
    where libname="SASHELP"
      and upcase(memname)="CARS" ;
 quit ;
 %put &mvariables  &mvariablesnum ;
```

SAS Log Results:

```
Make, Model, Type, Origin, DriveTrain, MSRP, Invoice,
EngineSize, Cylinders, Horsepower, MPG_City,
MPG_Highway, Weight, Wheelbase, Length          15
```

Cross-reference Listing

- Lafler (2013 and 2019) describes an approach that produces a cross-reference listing of all data sets (or tables) that contain a specific variable;
- Eliminates the time to review pages of PROC CONTENTS or PROC DATASETS output;
- Uses PROC SQL, PROC PRINT or your favorite PROC;
- Accesses the contents of Dictionary.COLUMNS table or the SASHELP.VCOLUMN view;
- Subsets columns: LIBNAME, MEMNAME, NAME, TYPE and LENGTH.

Cross-reference Listing – PROC SQL

Code:

```
PROC SQL ;
  SELECT LIBNAME, MEMNAME, NAME, TYPE, LENGTH
    FROM DICTIONARY.COLUMNS
      WHERE LIBNAME="SASHELP"
        AND UPCASE(NAME)="TYPE" ;
QUIT ;
```

Results:

Library Name	Member Name	Column Name	Column Type	Column Length
SASHELP	CARS	Type	char	8
SASHELP	COLUMN	TYPE	char	1
SASHELP	FEEDER	type	char	26
SASHELP	GCTYPE	type	char	14
SASHELP	MANAGE	TYPE	char	25
SASHELP	QUAKES	Type	char	10
SASHELP	REVHUB2	TYPE	char	16
SASHELP	SLKWXL	Type	char	32
SASHELP	SPRINGS	Type	char	9
SASHELP	SVRTDIST	Type	char	32
SASHELP	TABLE	TYPE	char	8
SASHELP	VCOLUMN	type	char	4
SASHELP	VDCTNRY	type	char	4
SASHELP	VPROMPT	type	char	4
SASHELP	VTITLE	type	char	1
SASHELP	_CMPIDX_	Type	char	8

Cross-reference Listing – PROC PRINT

Code:

```
PROC PRINT DATA=SASHELP.VCOLUMN NOOBS ;
  VAR LIBNAME MEMNAME NAME TYPE LENGTH ;
  WHERE LIBNAME="SASHELP"
    AND UPCASE(NAME)="TYPE" ;
RUN ;
```

Results:

Library Name	Member Name	Column Name	Column Type	Column Length
SASHELP	CARS	Type	char	8
SASHELP	COLUMN	TYPE	char	1
SASHELP	FEEDER	type	char	26
SASHELP	GCTYPE	type	char	14
SASHELP	MANAGE	TYPE	char	25
SASHELP	QUAKES	Type	char	10
SASHELP	REVHUB2	TYPE	char	16
SASHELP	SLKWXL	Type	char	32
SASHELP	SPRINGS	Type	char	9
SASHELP	SVRTDIST	Type	char	32
SASHELP	TABLE	TYPE	char	8
SASHELP	VCOLUMN	type	char	4
SASHELP	VDCTNRY	type	char	4
SASHELP	VPROMPT	type	char	4
SASHELP	VTITLE	type	char	1
SASHELP	_CMPIDX_	Type	char	8

Data-driven Programming Techniques Using CALL EXECUTE, User-defined Formats, and PROC SQL / Macros

Data-driven with CALL EXECUTE

- SAS users have a powerful DATA step routine called CALL EXECUTE that can be used for data-driven processing;
- The CALL EXECUTE routine accepts a single argument where the value can be a character-string or, when needed, a character expression containing SAS code elements to be executed after they are resolved;
- The CALL EXECUTE routine permits SAS statements and macro code to be stacked together and then executed.

Using CALL EXECUTE

The CALL EXECUTE argument can be specified with single or double quotes, dynamically generating SAS code for execution. Two steps are involved:

- ✓ **Create a Control Data Set;**
- ✓ **Process CALL EXECUTE statement(s).**

Step 1 – Create a Control Data Set

This CALL EXECUTE process creates a control data set containing six distinct car types (i.e., "Hybrid", "SUV", "Sedan", "Sports", "Truck", and "Wagon") represented as six separate observations.

```
data Cars_list ;  /* Control Data Set */
  input @1 Car_Type $6. ;
  datalines ;
Hybrid
SUV
Sedan
Sports
Truck
Wagon
;
run ;
```

Step 2 – Process Multiple CALL EXECUTEs

The next step reads the contents of the control data set populating the unique value for the Type variable in the individual CALL EXECUTE statements.

Note: The CATS function is used to strip blanks and concatenate multiple strings together.

Step 2 – Process Multiple CALL EXECUTEs

```
data _null_ ; /* Process in a DATA Step */
 set Cars_list ;
 call execute(CATS('ods Excel
                    file="d:/',Car_Type,'_Rpt.xlsx"
                    style=styles.barrettsblue
                 options(embedded_titles="yes");')) ;
 call execute(CAT('title ', Car_Type, ' Cars;')) ;
 call execute(CATS('proc freq
                     data=SASHELP.Cars(where=
                      (Type="',Car_Type,'"));')) ;
 call execute('tables Model;') ;
 call execute('run;') ;
 call execute('ods Excel close;') ;
run ;
```

CALL EXECUTE Results

Hybrid Cars

The FREQ Procedure

Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Civic Hybrid 4dr manual (gas/electric)	1	33.33	1	33.33
Insight 2dr (gas/electric)	1	33.33	2	66.67
Prius 4dr (gas/electric)	1	33.33	3	100.00

SUV Cars

The FREQ Procedure

Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent
4Runner SR5 V6	1	1.67	1	1.67
Ascender S	1	1.67	2	3.33
Aviator Ultimate	1	1.67	3	5.00
Aztekt	1	1.67	4	6.67
CR-V LX	1	1.67	5	8.33
Cayenne S	1	1.67	6	10.00
Discovery SE	1	1.67	7	11.67
Durango SLT	1	1.67	8	13.33
Element LX	1	1.67	9	15.00
Endeavor XLS	1	1.67	10	16.67

Sedan Cars

The FREQ Procedure

Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent
3.5 RL 4dr	1	0.38	1	0.38
3.5 RL w/Navigation 4dr	1	0.38	2	0.76
300M 4dr	1	0.38	3	1.15
300M Special Edition 4dr	1	0.38	4	1.53
325Ci 2dr	1	0.38	5	1.91
325Ci convertible 2dr	1	0.38	6	2.29
325i 4dr	1	0.38	7	2.67
325xi 4dr	1	0.38	8	3.05
330Ci 2dr	1	0.38	9	3.44
330Ci convertible 2dr	1	0.38	10	3.82
330i 4dr	1	0.38	11	4.20

Sports Cars

The FREQ Procedure

Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent
350Z Enthusiast convertible 2dr	1	2.04	1	2.04
350Z coupe 2dr	1	2.04	2	4.08
911 Carrera 4S coupe 2dr (convert)	1	2.04	3	6.12
911 Carrera convertible 2dr (coupe)	1	2.04	4	8.16
911 GT2 2dr	1	2.04	5	10.20
911 Targa coupe 2dr	1	2.04	6	12.24
Boxster S convertible 2dr	1	2.04	7	14.29
Boxster convertible 2dr	1	2.04	8	16.33
Celica GT-S 2dr	1	2.04	9	18.37

Truck Cars

The FREQ Procedure

Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Avalanche 1500	1	4.17	1	4.17
B2300 SX Regular Cab	1	4.17	2	8.33
B4000 SE Cab Plus	1	4.17	3	12.50
Baja	1	4.17	4	16.67
Canyon Z85 SL Regular Cab	1	4.17	5	20.83
Colorado Z85	1	4.17	6	25.00
Dakota Club Cab	1	4.17	7	29.17
Dakota Regular Cab	1	4.17	8	33.33
Escalade EXT	1	4.17	9	37.50

Wagon Cars

The FREQ Procedure

Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent
325xi Sport	1	3.33	1	3.33
9-5 Aero	1	3.33	2	6.67
A6 3.0 Avant Quattro	1	3.33	3	10.00
Aerio SX	1	3.33	4	13.33
C240	1	3.33	5	16.67

Data-driven User-defined Formats

- To prevent hard-coding VALUE clauses, custom-defined formats can be created dynamically from a SAS data set;
- This can be a more efficient approach than processing sort, merge, and join operations, by allowing leveraging data-driven processes;
- The FORMAT procedure is able to create informats and formats without specifying an INVALUE, PICTURE, or VALUE clause by using a SAS control data set as input.

Data-driven User-defined Formats

The control data set is specified with the CNTLIN option of PROC FORMAT. To start the process, the control data set must have the following variables:

- ✓ **FMTNAME - specifies the name of a character variable whose value is the format or informat name.**
- ✓ **START - specifies the name of a character variable that contains the value to be converted.**
- ✓ **LABEL - specifies the name of a character variable that contains the converted value.**

Step 1 – Create Control Table

A DATA step is specified with IF-THEN/ELSE logic to produce a control table with the required variables.

```
data control ;
  fmtname = "$Car_Origin" ;
  length label $13. ;
  input start $6. ;
  if start = “Asia"          then label = “Asian Cars" ;
  else if start = “Europe" then label = “European Cars";
  else if start = “USA"     then label = “USA Cars" ;
  output ;
  datalines ;
Asia
Europe
USA
;
run ;
```

Step 2 – Print Control Table Contents

The contents of the control data set is then displayed with the PRINT procedure.

```
proc print data=control noobs ;
  title ;
  var fmtname start label ;
run ;
```

Step 3 – Specify CNTLIN Option

Finally, the control data set is specified in the PROC FORMAT CNTLIN option.

```
proc format library=work cntlin=control ;
quit ;

proc print data=SASHELP.Cars noobs ;
  format Origin $Car_Origin. ;
run ;
```

User-defined Format Results

fmtname	start	label
$Car_Origin	Asia	Asian Cars
$Car_Origin	Europe	European Cars
$Car_Origin	USA	USA Cars

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	Engine Size	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
Acura	MDX	SUV	Asian Cars	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
Acura	RSX Type S 2dr	Sedan	Asian Cars	Front	$23,820	$21,761	2.0	4	200	24	31	2778	101	172
Acura	TSX 4dr	Sedan	Asian Cars	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
Acura	TL 4dr	Sedan	Asian Cars	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
Acura	3.5 RL 4dr	Sedan	Asian Cars	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
Acura	3.5 RL w/Navigation 4dr	Sedan	Asian Cars	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
Acura	NSX coupe 2dr manual S	Sports	Asian Cars	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
Audi	A4 1.8T 4dr	Sedan	European Cars	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	179
Audi	A41.8T convertible 2dr	Sedan	European Cars	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	180
Audi	A4 3.0 4dr	Sedan	European Cars	Front	$31,840	$28,846	3.0	6	220	20	28	3462	104	179
Audi	A4 3.0 Quattro 4dr manual	Sedan	European Cars	All	$33,430	$30,366	3.0	6	220	17	26	3583	104	179
Audi	A4 3.0 Quattro 4dr auto	Sedan	European Cars	All	$34,480	$31,388	3.0	6	220	18	25	3627	104	179
Audi	A6 3.0 4dr	Sedan	European Cars	Front	$36,640	$33,129	3.0	6	220	20	27	3561	109	192
Audi	A6 3.0 Quattro 4dr	Sedan	European Cars	All	$39,640	$35,992	3.0	6	220	18	25	3880	109	192
Audi	A4 3.0 convertible 2dr	Sedan	European Cars	Front	$42,490	$38,325	3.0	6	220	20	27	3814	105	180
Audi	A4 3.0 Quattro convertible 2dr	Sedan	European Cars	All	$44,240	$40,075	3.0	6	220	18	25	4013	105	180
Audi	A6 2.7 Turbo Quattro 4dr	Sedan	European Cars	All	$42,840	$38,840	2.7	6	250	18	25	3836	109	192

Creating Multiple Data Sets and Output

- Lafler (2018 and 2019) offers ways to create multiple data sets (or tables) and output;
- Traditional (non Data-driven) approach using the DATA step with the OUTPUT statement;
- Data-driven approach using PROC SQL and the Macro language by iterating through a process to create multiple data sets (or tables) and output.

Multiple Data Sets – Non Data-driven

Code:

```
Data Hybrid
     SUV
     Sedan
     Sports
     Truck
     Wagon ;
  set sashelp.cars ;
  if type = “Hybrid”      then output Hybrid ;
  else if type = “SUV”    then output SUV ;
  else if type = “Sedan”  then output Sedan ;
  else if type = “Sports” then output Sports ;
  else if type = “Truck”  then output Trucks ;
  else if type = “Wagon”  then output Wagon ;
Run ;
```

Multiple Data Sets – Non Data-driven

#	Name	Member Type	File Size	Last Modified
1	HYBRID	DATA	128KB	04/03/2019 08:24:36
2	SEDAN	DATA	128KB	04/03/2019 08:24:37
3	SPORTS	DATA	128KB	04/03/2019 08:24:37
4	SUV	DATA	128KB	04/03/2019 08:24:37
5	TRUCK	DATA	128KB	04/03/2019 08:24:37
6	WAGON	DATA	128KB	04/03/2019 08:24:37

Library Name	Member Name	Member Type	DBMS Member Type	Data Set Label	Data Set Type	Date Created	Date Modified	Number of Physical Observations	Observation Length	Number of Variables
WORK	HYBRID	DATA			DATA	03APR19:08:24:36	03APR19:08:24:36	3	152	15
WORK	SEDAN	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	262	152	15
WORK	SPORTS	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	49	152	15
WORK	SUV	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	60	152	15
WORK	TRUCK	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	24	152	15
WORK	WAGON	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	30	152	15

Data-driven Approach

- A data-driven approach is used with PROC SQL and the macro language to perform iterative processes to create results and output;
- The SQL procedure and the macro language are two robust tools found in Base SAS software;
- Triggered by calling a macro to reduce coding requirements, the process uses the Macro language, PROC SQL, the ODS Excel destination, and PROC FREQ to send output (results) to Excel.

Multiple Data Sets – Data-driven

Code:

```
options symbolgen ;

%macro multidatasets ;
  proc sql noprint ;
   select count(distinct type)
    into :mtype_cnt  /* number of unique types */
     from SASHELP.cars ;
   select distinct type
    into :mtype_lst separated by "~“ /* values */
     from SASHELP.CARS ;
  quit ;
```

Multiple Data Sets – Data-driven

Code (continued):

```
  %do i = 1 %to &mtype_cnt ;
      proc sql ;
       create table %scan(&mtype_lst,&i,'~')
        select *
         from SASHELP.cars
          where type="%scan(&mtype_lst,&i,'~')" ;
      quit ;
  %end ;

%mend multidatasets ;

%multidatasets ;
```

Multiple Data Sets – Data-driven Results

#	Name	Member Type	File Size	Last Modified
1	HYBRID	DATA	128KB	04/03/2019 08:24:36
2	SEDAN	DATA	128KB	04/03/2019 08:24:37
3	SPORTS	DATA	128KB	04/03/2019 08:24:37
4	SUV	DATA	128KB	04/03/2019 08:24:37
5	TRUCK	DATA	128KB	04/03/2019 08:24:37
6	WAGON	DATA	128KB	04/03/2019 08:24:37

Library Name	Member Name	Member Type	DBMS Member Type	Data Set Label	Data Set Type	Date Created	Date Modified	Number of Physical Observations	Observation Length	Number of Variables
WORK	HYBRID	DATA			DATA	03APR19:08:24:36	03APR19:08:24:36	3	152	15
WORK	SEDAN	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	262	152	15
WORK	SPORTS	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	49	152	15
WORK	SUV	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	60	152	15
WORK	TRUCK	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	24	152	15
WORK	WAGON	DATA			DATA	03APR19:08:24:37	03APR19:08:24:37	30	152	15

Another Data-driven Example

- As before, PROC SQL and the macro language are used to create a macro routine that performs an iterative process to produce results and output;
- Create a single-value (or aggregate) macro variable and a value-list (or array of values) macro variable;
- Triggered by calling a macro to reduce coding requirements, the process uses the Macro language, PROC SQL, the ODS Excel destination, and PROC FREQ to send output (results) to Excel.

Step 1 – Use PROC FORMAT – Traffic Lights

```
PROC SORT DATA=SASHELP.CARS
            OUT=WORK.CARS_SORTED ;
  BY Origin MSRP ;
RUN ;

PROC FORMAT ;
  Value MSRPFmt LOW - < 20000 = 'Green'
                20000 - < 35000 = 'Blue'
                35000 - < 50000 = 'Orange'
                50000 - HIGH    = 'Red' ;
RUN ;
```

Step 2 – Create Macro Variables with SQL

```
%macro multiExcelfiles ;
  proc sql noprint ;
   select count(distinct Origin)
    into :morigin_cnt /* number of unique origins */
     from WORK.Cars_Sorted ;
   select distinct Origin
    into :morigin_lst separated by "~" /* Values */
     from WORK.Cars_Sorted ;
  quit ;
```

Step 3 – Process Macro Variables

```
%do i=1 %to &morigin_cnt ;
    ods Excel file="d:/%SCAN(&morigin_lst,&i,~).xlsx" ;
     proc report data=SASHELP.Cars NOWINDOWS
          STYLE(Header)={BackGround=Blue ForeGround=White
                         Font=(Arial,10pt,Bold)} ;
       where Origin="%SCAN(&morigin_lst,&i,~)" ;
       COLUMNS Origin Make Type Model MSRP ;
       DEFINE MSRP / ANALYSIS 'Vehicle MSRP' WIDTH=8
         STYLE(Column)=[FontWeight=bold BackGround=MSRPFmt.] ;
       COMPUTE MSRP ;
        CALL DEFINE(_COL_,"STYLE","STYLE={ForeGround=White}") ;
       ENDCOMP ;
     run ;
     ods Excel close ;
  %end ;
%mend multiExcelfiles ;
%multiExcelfiles ;
```

Adding Color to Excel – Results

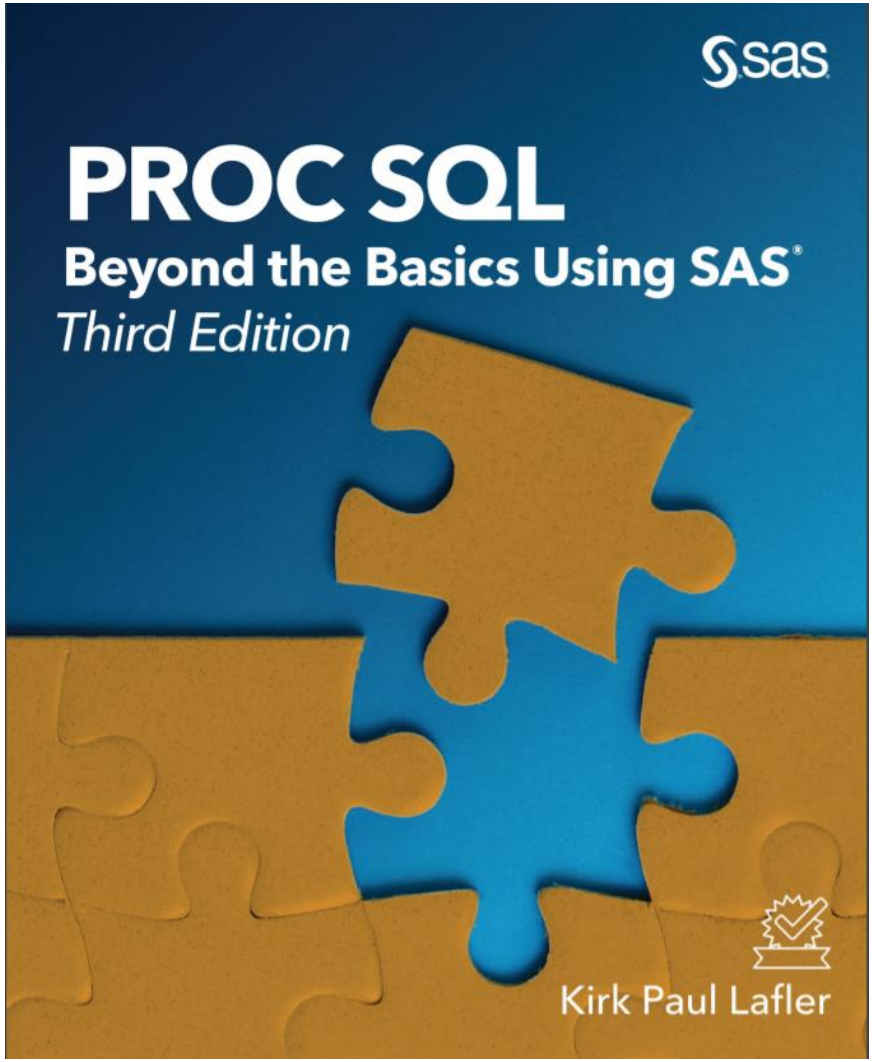
Origin	Make	Type	Model	Vehicle MSRP
Asia	Kia	Wagon	Rio Cinco	$11,905
Asia	Toyota	Truck	Tacoma	$12,800
Asia	Scion	Wagon	xB	$14,165
Asia	Mazda	Truck	B2300 SX Regular Cab	$14,840
Asia	Toyota	Truck	Tundra Regular Cab V6	$16,495
Asia	Suzuki	Wagon	Aerio SX	$16,497
Asia	Toyota	Wagon	Matrix XR	$16,695
Asia	Mitsubishi	Wagon	Lancer Sportback LS	$17,495
Asia	Nissan	Truck	Frontier King Cab XE V6	$19,479
Asia	Subaru	Wagon	Forester X	$21,445
Asia	Mazda	Truck	B4000 SE Cab Plus	$22,350
Asia	Subaru	Wagon	Outback	$23,895
Asia	Subaru	Truck	Baja	$24,520
Asia	Toyota	Truck	Tundra Access Cab V6 SR5	$25,935
Asia	Nissan	Truck	Titan King Cab XE	$26,650
Asia	Nissan	Wagon	Murano SL	$28,739
Asia	Lexus	Wagon	IS 300 SportCross	$32,455
Asia	Infiniti	Wagon	FX35	$34,895
Asia	Infiniti	Wagon	FX45	$36,395
Europe	Volkswagen	Wagon	Jetta GL	$19,005
Europe	Volkswagen	Wagon	Passat GLS 1.8T	$24,955
Europe	Volvo	Wagon	V40	$26,135
Europe	BMW	Wagon	325xi Sport	$32,845
Europe	Mercedes-Benz	Wagon	C240	$33,780
Europe	Volvo	Wagon	XC70	$35,145
Europe	Volkswagen	Wagon	Passat W8	$40,235
Europe	Audi	Wagon	A6 3.0 Avant Quattro	$40,840
Europe	Saab	Wagon	9-5 Aero	$40,845
Europe	Audi	Wagon	S4 Avant Quattro	$49,090
Europe	Mercedes-Benz	Wagon	E320	$50,670
Europe	Mercedes-Benz	Wagon	E500	$60,670
USA	Ford	Truck	Ranger 2.3 XL Regular Cab	$14,385
USA	GMC	Truck	Canyon Z85 SL Regular Cab	$16,530
USA	Pontiac	Wagon	Vibe	$17,045
USA	Ford	Wagon	Focus ZTW	$17,475
USA	Dodge	Truck	Dakota Regular Cab	$17,630
USA	Chevrolet	Truck	Colorado Z85	$18,760
USA	Dodge	Truck	Ram 1500 Regular Cab ST	$20,215
USA	Dodge	Truck	Dakota Club Cab	$20,300
USA	Chevrolet	Truck	Silverado 1500 Regular Cab	$20,310
USA	Ford	Truck	F-150 Regular Cab XL	$22,010
USA	Chevrolet	Wagon	Malibu Maxx LS	$22,225
USA	Ford	Wagon	Taurus SE	$22,290
USA	Mercury	Wagon	Sable GS	$22,595
USA	Saturn	Wagon	L300 2	$23,560
USA	GMC	Truck	Sonoma Crew Cab	$25,395
USA	GMC	Truck	Sierra Extended Cab 1500	$25,717
USA	GMC	Truck	Sierra HD 2500	$29,322
USA	Chrysler	Wagon	Pacifica	$31,230
USA	Ford	Truck	F-150 Supercab Lariat	$33,540
USA	Chevrolet	Truck	Avalanche 1500	$36,100
USA	Chevrolet	Truck	Silverado SS	$40,340
USA	Chevrolet	Truck	SSR	$41,995
USA	Cadillac	Truck	Escalade EXT	$52,975

Conclusion

Popular Programming Paradigms

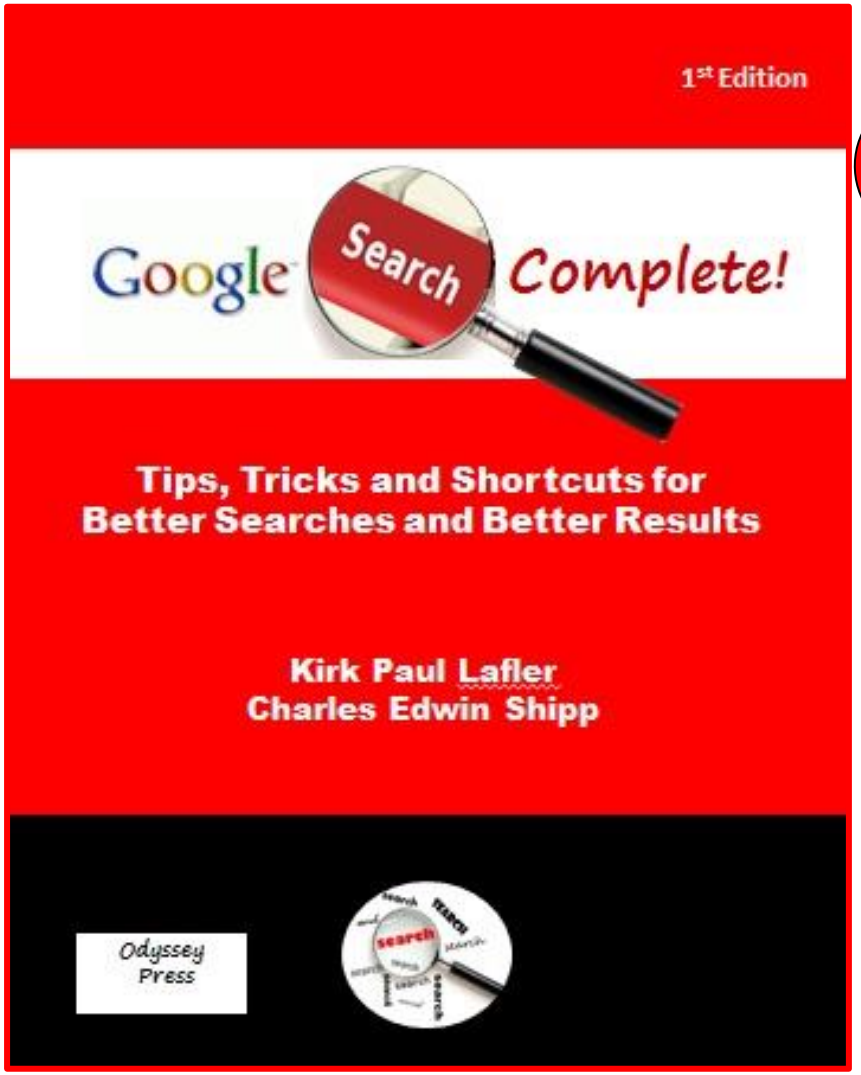
Data-driven Programming Techniques Using Metadata Sources

Data-driven Programming Techniques Using CALL EXECUTE, User-defined Formats, and PROC SQL / Macros



An SQL Book with "under the hood" details, explanations and lots of examples

Available from SAS Press and other online retailers !



A Google Search Book the Pros use for Better Searches and Better Results!

Available on www.Amazon.com!

Thank You for Attending!

Questions?

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