FAA

Fractal Academy of Analytics

SAS Training

Features and Understanding
of
PROC SQL

Contents

Structured Query Language (SQL)

- Structured Query Language
- Developed by IBM in the early 1970’s
- From the 70’s to the late 80’s there were different types of SQL, based on different databases.
- In 1986 the first unified SQL standard (SQL-86) was created.
- In 1987 database interface for SQL was added to the Version 6 Base SAS package
- A “language within a language”

Features and Advantages

- SAS looks at a dataset one record at a time, using an implied loop that moves from the first record to the last
- Because of this difference SQL can easily do a few things that are more difficult to do in SAS
- SQL commands are available for creating tables, changing table structures, changing values in tables, functions and more…

- Combined functionality
- Faster for smaller tables
- SQL code is more portable for non-SAS applications
- Not require presorting
- Not require common variable names to join on. (need same type , length)

Comparison

- Proc SQL helps to solve the different Data step solution into one like Proc Sort ,Proc Summary and Proc Print.
- One very distinct and important difference between PROC SQL and the DATA step is that the former cannot create tables from non-relational external data sources while the later can.
- One highly touted benefit of PROC SQL is its ability to process unsorted data and create tables in a sorted fashion.

Function

- PROC SQL supports all the functions available to the SAS DATA step that can be used in a proc Sql select statement
- Because of how SQL handles a dataset, these functions work over the entire dataset
- Common Functions:
  - COUNT
  - DISTINCT
  - MAX
  - MIN
  - SUM
  - AVG
  - VAR
  - STD
  - STDERR
  - NMISS
  - RANGE
  - SUBSTR
  - LENGTH
  - UPPER
  - LOWER
  - CONCAT
  - ROUND
  - MOD

- PROC SQL does not support LAG, DIF, and SOUND functions.

SQL DML and DDL

- SELECT
- WHERE
- Alias
- DISTINCT
- ORDER BY
- IN
- BETWEEN
- LIKE
- GROUP BY
- HAVING
- JOINS
- INSERT,UPDATE,DELETE

- CREATE
- DROP
- ALTER
- Modify

What SQL can do?

- Selecting
- Ordering/sorting
- Subsetting
- Restructuring
- Creating table/view
- Joining/Merging
- Summarizing
- Transforming variables
- Editing

Proc Sql Intro - Terminology

SAS Data Step	Proc SQL
Dataset	Table
Variables	Column
Observation	Row
Merge	Join
Append	Union

Syntax

```
PROC SQL;
    CREATE TABLE tablename as
     SELECT column(s)
       FROM table-name | view-name
         WHERE expression
            GROUP BY column(s)
              HAVING expression
                ORDER BY column(s);
QUIT;
```

Data Tablename

Keep Column1,column2

Set library.table

Where expression

Proc Sort;
By column 1;
Run;

Dataset as Books

Title	Author	ISBN	Price	Quantity
The Little SAS Book	Delwiche	1590473337	365	35
SAS Survival Handbook	Wiseman	60578793	450	40
SAS for Dummies	McDaniel	471788325	780	14
Learning SAS by Example	Cody	1599941651	469	12
Output Delivery System	Haworth	1590473787	1,123	10
SAS Functions by Example	Cody	1580255787	390	24
Annotate: Simply the Basics	Carpenter	1891957112	234	11
SAS Programming Shortcuts	Aster	1580259243	457	5
Survival Analysis Using SAS	Allison	1590478827	235	7
Longitudinal Data and SAS	Cody	158025859X	1,100	9
SAS Macro Programming	Burlew	155544279X	1,345	15

Selecting Data

```
PROC SQL;
SELECT count (DISTINCT Author)
 FROM library. Books;
QUIT;
```

Distinct ~ No Duplicate values

- The simplest SQL code, need 3 statements
- By default, it will print the resultant query, use NOPRINT option to suppress this feature.
- Begin with ***PROC SQL***, end with ***QUIT****;* not ***RUN;***
- Need at least one ***SELECT… FROM*** statement
- *DISTINCT* is an option that removes duplicate rows

Ordering/Sorting Data

```
PROC SQL ;
 SELECT *
  FROM Library. Books
  ORDER BY ISBN;
QUIT;
```

* = all variables

- Remember the placement of the SAS statements has no effect; so we can put the middle statement into 3 lines
- SELECT * means we select all variables from dataset Library. Books
- Put ORDER BY after FROM.
- We sort the data by variable “ISBN”

Subsetting Data
- Character searching in WHERE

```
PROC SQL;
 SELECT Title, Author, ISBN
  FROM Library. Books
  WHERE Author CONTAINS ‘Cody';
QUIT;
```

- Use comma **(,)** to select multiple variables
- CONTAINS in WHERE statement only for character variables
- Also try WHERE UPCASE(Author) LIKE '% Cody %';
- Use wildcard char. Percent sign (%) with LIKE operator`

Case Logic - reassigning/recategorize

```
PROC SQL;
SELECT Title, Author,
        CASE Author
             WHEN ‘Cody'  THEN ‘General'   ELSE  'Other'
        END AS level
 FROM Library. Books;
QUIT;
```

- The order of each statement is important
- CASE …END AS should in between SELECT and FROM
- Note there is , after the variables you want to select
- Use WHEN … THEN ELSE… to redefine variables
- Rename variable from “Author” to “level”

Case Logic
-Sum/Count

```
PROC SQL;
SELECT
        SUM (CASE WHEN Author= 'G' THEN 1 END) as General,
        SUM( CASE WHEN Author=‘G’ THEN Price END) as Price,
         Count (*) as Books
    FROM Library. Books
Group by Author;
QUIT;
```

The Count () function returns the number of rows as defined by the Group Statement

Creating New Data - Create Table

```
PROC SQL;
 CREATE TABLE New as
 SELECT Title, Author, Price, ISBN
  FROM Library. Books
  WHERE Author CONTAINS ‘Haworth';
QUIT;
```

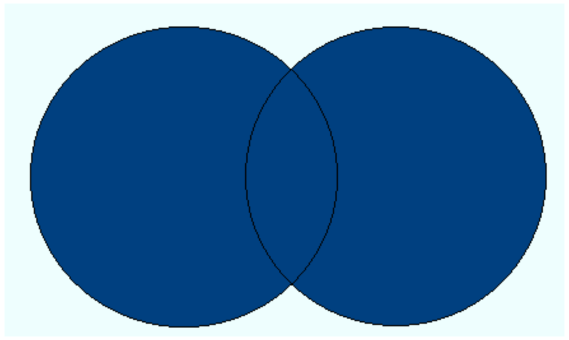
- CREATE TABLE … AS can always be in front of SELECT … FROM statement to build a sas file.
- In SELECT, the results of a query are converted to an output object (printing). Query results can also be stored as data. The CREATE TABLE statement creates a table with the results of a query.

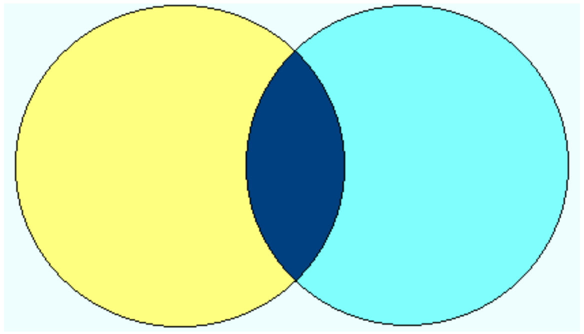
Creating New Data
- Create View

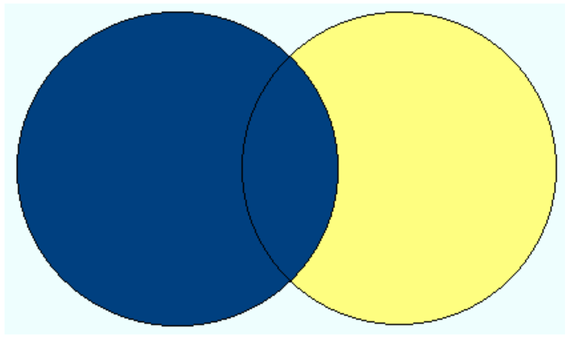
```
PROC SQL;
 CREATE VIEW G_MOVIES as
          SELECT Title, Author, ISBN, Price
          FROM Library. Books
    WHERE Price = 235
          ORDER BY Price;
 SELECT  *  FROM G_MOVIES;
QUIT;
```

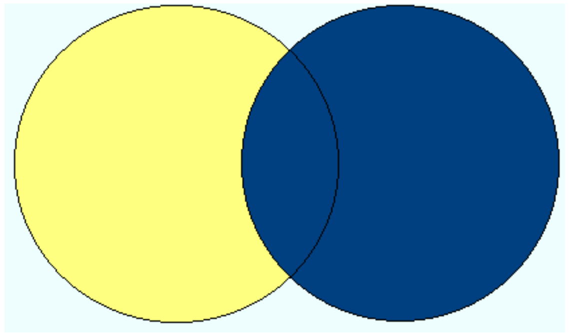
- First step-creating a view, no output is produced; then display the desired output results
- Use ; to separate two block of code inside of proc sql
- When a table is created, the query is executed and the resulting data is stored in a file. When a view is created, the query itself is stored in the file. The data is not accessed at all in the process of creating a view.

Join Tables (Merge Dataset)
Cartesian Join


Full Join


Inner Join


Left Join


Right Join

Cartesian Product Join

- When two or more tables are specified in the FROM clause of a SELECT query without a corresponding WHERE clause expression, a special type of join is created. The Cartesian product (or Cross join) represents all possible combinations of rows and columns from the joined tables. To be exact, Cartesian product joins represent the sum of the number of columns of the input tables plus the product of the number of rows of the input tables. Essentially the Cartesian product contains *m * n rows, where m is the number of rows in the first table and n is the number of rows in the second table. Put another way, it represents each row from the first table matched with each possible row from the second table, and so on and so forth. All other types of joins are classified as subsets of Cartesian products essentially being created by deriving the Cartesian product and then excluding rows that fail the specified WHERE clause expression.*

Proc Sql Syntax – Joining Table

```
PROC SQL;
 CREATE TABLE tablename as
 SELECT
        alias1.Column 1,
         alias2.column 2
  FROM library.table1 as alias1
           [Inner|Left|Right|Full] Join
           library.table2 as alias2
  on join clause;
QUIT;
```

Specify how the two tables are joined

Sort of the by statement in a SAS Merge

Cartesian Product – Joining Table

```
PROC SQL;
 CREATE TABLE tablename as
 SELECT
        alias1.Column 1,
         alias2.column 2
  FROM library.table1 as alias1
            library.table2 as alias2
  on join clause;
QUIT;
```

Failing to specify a join clause will create a Cartesian product

i.e every row in table1 will be matched to every row in table 2

Dataset as Order Date

ISBN	Order Date
1590473337	1/23/2011
60578793	7/12/2011
471788325	12/12/2010
1599941651	11/10/2010
1590473787	11/5/2007
1580255787	8/30/2008
1891957112	3/25/2009
1580259243	5/9/2008
1891957112	1/23/2009
158025859X	7/7/2011
158025859X	12/3/2010

Cartesian Join

```
PROC SQL;
 SELECT *
  FROM Library.Books,
           Library.Order_Date;
QUIT;
```

- Terminology: Join (Merge) datasets (tables)
- No prior sorting required – one advantage over DATA MERGE
- Use comma (,) to separate two datasets in FROM
- Without WHERE, all possible combinations of rows from each tables is produced, all columns are included
- Turn on the HTML result option for better display:

Tool/Options/Preferences…/Results/ check Create HTML/OK

Inner Join using WHERE

```
PROC SQL;
Create table New_data as
  SELECT B.Author, B.Price, O.ISBN, O.Order_date
  FROM Library.Books as B, Library.Order_Date as O
    WHERE B.ISBN  =  O.ISBN;
QUIT;
```

- Short-cut for table names
  - Can be used in SELECT and WHERE statements
  - Need to be declared in FROM statement

Dataset as Delivery Date

Book_Num	Delivery Date
1590473337	1/30/2011
60578793	7/19/2011
471788325	12/19/2010
1599941651	11/19/2010
1590473787	11/12/2007
1580255787	9/7/2008
1891957112	3/31/2009
1580259243	5/17/2008
1891957112	1/30/2009
158025859X	7/14/2011
158025859X	12/10/2010

Join three tables

```
PROC SQL;
  SELECT B.Author, B.Title,
                  O.Order_Date,
                   D.Book_Num,
                    D.Delivery_Date
FROM Library.Books as B,
                Library.Order_Date as O,
                Library.Delivery_Date as D
     WHERE B.ISBN = O.ISBN  AND D.Book_Num=O.ISBN;
QUIT;
```

- Use AND in WHERE statement to specify two matching conditions
- Produce rows that satisfies all the conditions
- Can join up to 32 tables in one SQL code

Inner Joins using ON

```
PROC SQL;
  SELECT B.Author, B.Title, B.ISBN, B.Price
  FROM Library. Books as B
           INNER JOIN
  Library. Order_Date as O
          ON B.ISBN  =  O.ISBN;
QUIT;
```

- Same result as using where
  - WHERE is used to select rows from inner joins
  - ON is used to select rows from outer or inner

Left Outer Joins

```
PROC SQL;
  SELECT B.Author, B.Title, B.ISBN, B.Price
  FROM Library.Books as B
          LEFT JOIN
  Library.Order_Date as O
         ON B.ISBN  =  O.ISBN;
QUIT;
```

- Resulting output contains all rows for which the SQL expression, referenced in the ON clause, matches both tables and all rows from LEFT table (Books) that did not match any row in the right (Order_date) table.
- Essentially the rows from LEFT table are preserved and captured exactly as they stored in the table itself, regardless if a match exists.
- Need to specify a table name for the matching variable in SELECT

Right Outer Joins

```
PROC SQL;
  SELECT B.Author, B.Title, B.ISBN, B.Price
  FROM Library.Books as B
           RIGHT JOIN
  Library. Order_Date as O
          ON B.ISBN  =  O.ISBN;
QUIT;
```

- Resulting output contains all rows for which the SQL expression, referenced in the ON clause, matches both tables and all rows from RIGHT table (Order_Date) that did not match any row in the right (Books) table.
- Essentially the rows from RIGHT table are preserved and captured exactly as they stored in the table itself, regardless if a match exists.

Outer Union

```
PROC SQL;
  SELECT * FROM Library.Books
  OUTER UNION
  SELECT * FROM Library.Order_Date;
QUIT;
```

- SQL performs OUTER UNION, similar to DATA steps with a SET statement to Concatenate datasets.
- The result contains all the rows produced by the first table-expression followed by all the row produced by the second table-expression.

Except option

```
PROC SQL;
   SELECT * FROM Library.Books
     EXCEPT
       SELECT * FROM Library.Order_Date;
QUIT;
```

- The EXCEPT operator produces (from the first table-expression) an output table that has unique rows that are not in the second table-expression

Self Merging

```
PROC SQL;
   SELECT
            B.* , A.Price as Price1
     FROM      Library.Books as B, Library.Books as A
      WHERE
               B.Author=A.Auhtor;
 QUIT;
```

- It is used like the Lag function
- It is also called as reflexive join
- It is used to show comparative relationship between values in a table
- When we merge at that time the Primary key should be of same name and the other variable should be renamed in the other dataset

Transforming Data
- Creating new Variables

```
/*Creating new variables*/

PROC SQL;
 SELECT Title, Author, ISBN, Price/50 as Dollar_Price
 FROM Library.Books;
QUIT;
```

- You can create new variables within SELECT statement, the name of new variable follows after AS.
- Note the order of the express is reversed

Phonetic Matching

```
PROC SQL;
   SELECT Author, ISBN, Price
     FROM Library.Books
       WHERE Author =* “McDaniel”;
QUIT;
```

- Not Technically a Function.
- Finding name that sounds alike or have spelling Variations
- Searches and selects character data based on two expressions: the search value and the matched value.

Finding Patterns in a String

```
PROC SQL;
       SELECT Author, ISBN, Price
        FROM Library.Books
        WHERE category LIKE ‘D%’;
QUIT;
```

- The % acts as a wildcard character representing any number of characters, including any combination of upper or lower case characters.

Transforming Data
- Summarizing Data using SQL functions

```
PROC SQL;
 SELECT *,
            COUNT(Title)  AS notitle,
            MAX(Price)      AS Expensive,
            MIN(Price)       AS Cheapest,
            SUM(Price)   AS Total_Cost,
            NMISS(Author) AS nomissing
 FROM Library.Books
 GROUP BY Author;
QUIT;
```

- Simple summarization functions available
- All function can be operated in Groups
- Re-merging summary statistics with Original data

Editing Data
– Insert observations.

```
PROC SQL NOPRINT;
INSERT INTO MFE.CUSTOMERS(a,b)
 VALUES(1 'Peng‘, 2 ’rid’,3 ‘sam’);
INSERT INTO MFE.CUSTOMERS
 SET Cust_no=2,Name='Sasha';
QUIT;
```

- There are two ways of inserting observations into a table. Data type should be the same.
- VALUES( ) new values are separated by space.
- SET column name = newly assigned values, delimited by commas.

Editing Data

– Deleting rows and Dropping columns

```
/*Deleting rows*/
PROC SQL;

DELETE
 FROM Library.Books
 WHERE Price LE 100;
QUIT;
```

```
/*Dropping variables*/
PROC SQL;
 CREATE TABLE NEW
    (DROP=ISBN) AS
 SELECT *
  FROM Library.Books;
QUIT;
```

- Deleting columns can be done in SELECT or in DROP on created table

Editing Data

– Renaming Rows and No. of Observation

```
/*Renaming rows*/
PROC SQL;
Create table rename
(rename =(ISBN=Number)) as
    FROM Library.Books
  WHERE Price LE 100;
QUIT;
```

```
/*Observation  selection*/
PROC SQL;
 CREATE TABLE Observation
      as
 SELECT *
  FROM Library.Books (obs=5);
QUIT;
```

- Renaming columns can be done in CREATE Statement
- Selection of Number of Observation is done in the FROM Statement

Editing Data
– Update observations

```
/*Updating Observation*/
PROC SQL NOPRINT;
UPDATE Library.Books
  SET Author=' Cody '
  WHERE ISBN='155544279X' ;
QUIT;
```

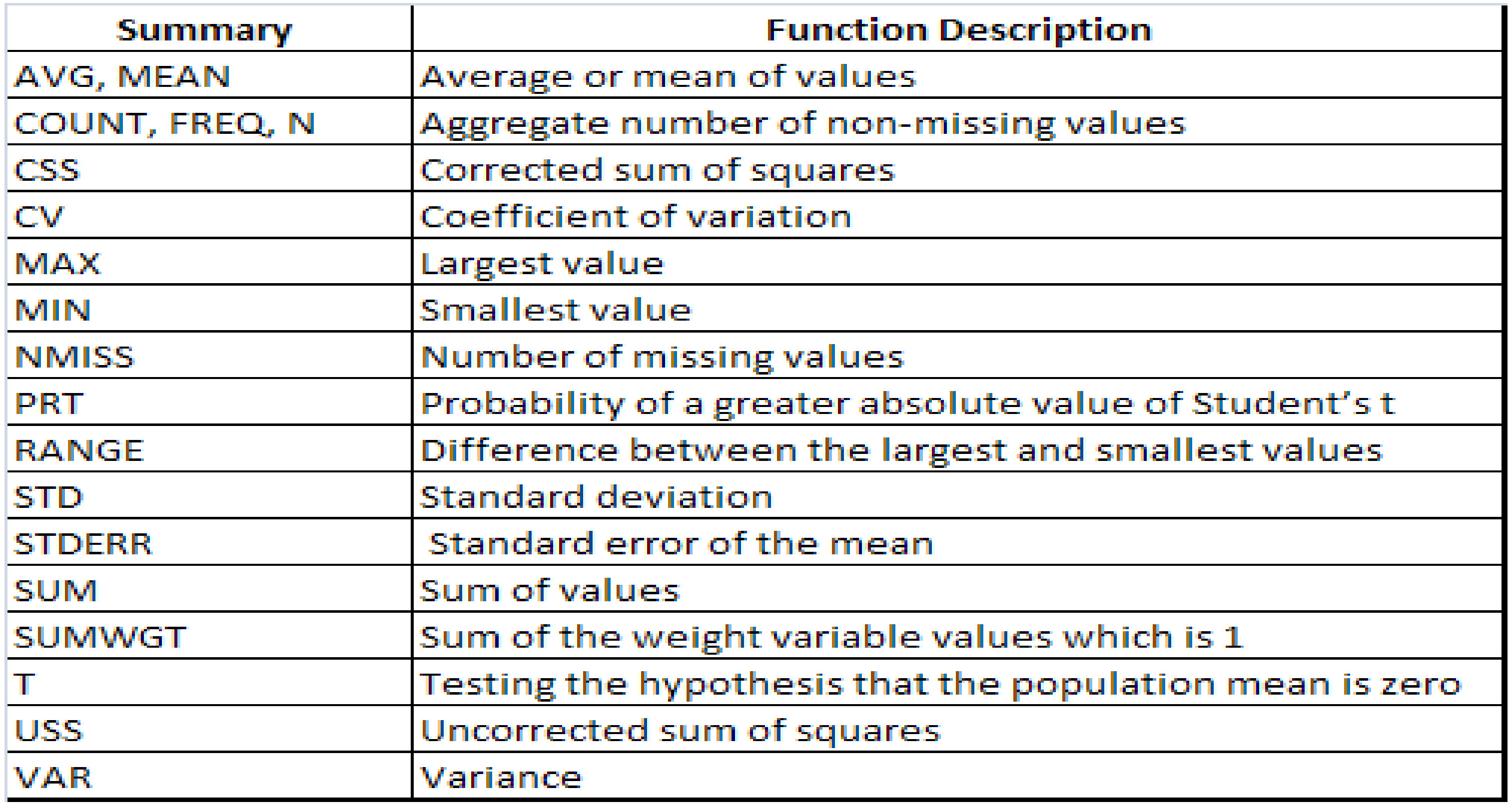
- UPDATE … SET… WHERE
- Find the observation and set new value
- If more than one observations satisfies the condition, all are updated with the new data in SET statement

Identifying Duplicates

```
PROC SQL;
   CREATE TABLE Author as
      SELECT Author,
                     Sum(Price) as Price
         FROM Library.Books
           GROUP BY Author
            HAVING COUNT(*) GE 2
            ORDER BY Price ;
QUIT;
```

Summarizing Data

- It provides a number of useful summary (or aggregate) functions to help perform calculations, descriptive statistics, and other aggregating operations in a SELECT statement or HAVING clause.



Summary	Function Description
AVG, MEAN	Average or mean of values
COUNT, FREQ, N	Aggregate number of non-missing values
CSS	Corrected sum of squares
CV	Coefficient of variation
MAX	Largest value
MIN	Smallest value
NMISS	Number of missing values
PRT	Probability of a greater absolute value of Student's t
RANGE	Difference between the largest and smallest values
STD	Standard deviation
STDERR	Standard error of the mean
SUM	Sum of values
SUMWGT	Sum of the weight variable values which is 1
T	Testing the hypothesis that the population mean is zero
USS	Uncorrected sum of squares
VAR	Variance

Correlated Sub-Query

```
PROC SQL;
    CREATE TABLE Corr_Query AS
      SELECT DISTINCT Author, Price
        FROM Library.Books as B
          WHERE ‘1590473337’ IN
            (SELECT Order_Date
              FROM Library.Order_Date as O
                 WHERE B.ISBN=O.ISBN)
                    ORDER BY Price;
QUIT;
```

Non-Correlated Sub-Query

```
PROC SQL;
CREATE TABLE NON_CORR as
      SELECT Author,
                 Avg(Price) as Avg_Price
                      FORMAT = dollar11.2
        FROM Library.Books
        GROUP BY Author
        HAVING Avg(Price) >
                  (SELECT Avg(Price)
                   FROM Library.Books);
    QUIT;
```

Using calculated field in SQL

```
PROC SQL;
CREATE TABLE new as
SELECT Author ,Price  ,(Price*10) as Value
FROM Library.Books
WHERE calculated Value > 5000
GROUP by Author
HAVING Mean(Value) gt 54;
QUIT;
```

Validation of Syntax

```
PROC SQL;
       VALIDATE
      SELECT *
          FROM Library.Books
              WHERE Price= 780;
QUIT;
```

- Help in Troubleshooting and debugging the SQL Queries.
- It is Specified in Conjunction with a SELECT Statement.
- The appropriate message is displayed on the SAS log to indicate whether coding problems exist.

Validation of Syntax

```
PROC SQL noexec;
     SELECT *
        FROM Library.Book
           WHERE Price= 780;
QUIT;
```

- Help in Troubleshooting and debugging the SQL Queries.
- It is specified in the Procedure Step
- The appropriate message is displayed on the SAS log to indicate whether coding problems exist.

Display the Contents of Dataset

```
PROC SQL;
  Describe table Library.Books;
QUIT;
```

- Help in Displaying the Variables Name with format and Informat of the Variables
- Just like the Proc Contents
- Display the result only in Log

Multiple Dataset in Single Proc Sql

```
PROC SQL;
 Create table Obs as
  SELECT *
   FROM Library.Books (obs=4);

   Create table Author as
   Select Author
   from Library.Books
   where Author='Cody';

QUIT;
```

Errors

- Syntax error, expecting one of the following: !, !!, &, (, *, **, +, ',', -, /, <, <=, <>, =, >, >=, ?, AND, BETWEEN, CONTAINS, EQ, FROM, GE, GT, LE, LIKE, LT, NE, OR, ^=, |, ||, ~=.
- ERROR: Libname LIBRARY is not assigned.
- ERROR: File WORK.BOOKS.DATA does not exist.
- WARNING: This SAS global statement is not supported in PROC SQL. It has been ignored.
- ERROR : Syntax error, statement will be ignored.
- WARNING: Data too long for column "COMMENT"; truncated to 124 characters to fit.
- WARNING: Variable N_Time already exists on file WORK.LD50_PARMS.

Thank You