PROC SQL

Proc SQL is part of Base SAS and can combine some features of data steps and procs.

SQL stands for Structured Query Language, and is considered the standard language for relational databases. Database management systems that use SQL include Access, Ingres, Oracle, Sybase, Microsoft SQL Server, etc

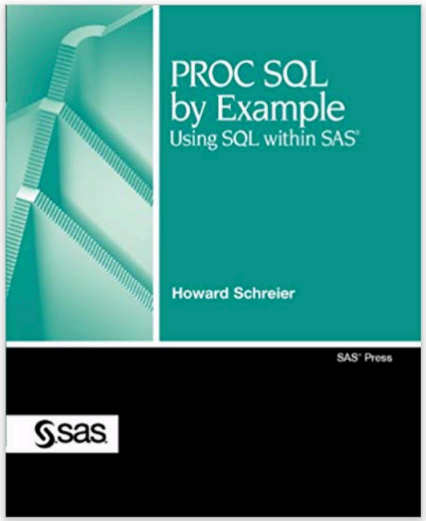
There is a whole book on PROC SQL (300 pages):

PROC SQL by Example: Using SQL within SAS 1st Edition

by Howard Schreier (Author)

5 customer reviews

Look inside



ISBN-13: 978-1599942971

Perfect Paperback
$31.11 - $44.62

Other Sellers
from $31.11

Buy used

Buy new

In Stock.
Ships from and sold by Amazon.com. Gift-wrap available.

Note: Available at a lower price from other sellers that may not offer

Want it Friday, Nov. 30? Order within **20 hrs 39 mins** and choose **On**
Details

PROC SQL

Apparently, PROC SQL can replicate much of the functionality of the datastep. Proc SQL has a different flavor than the rest of SAS so feels like a different language. One approach is to do almost all analysis using PROC SQL. Another is to go back and forth, such as reading in data using a traditional data step, and then use PROC SQL to manipulate the data. The syntax in PROC SQL should be similar to SQL as implemented in other languages, and so is a good way to learn SQL.

PROC SQL

One difference between PROC SQL and the data step is that items in a series are usually separated by spaces in the data step but by commas in SQL. There are also terminological differences:

Realm	Corresponding Terms		
SAS	SAS data file	Observation	Variable
SQL	Table	Row	Column
Data processing	File	Record	Field
Relational database theory	Relation	Tuple	Attribute

PROC SQL

Just as in the data step, PROC SQL runs statements that are terminated by semi-colons. However, the SQL block is terminated by `QUIT;` instead of `run;`

Technically, SAS is considered a **procedural** language, meaning you tell the computer what to do. SQL is considered **declarative**, meaning you tell the computer what to produce, and the software determines what algorithm will produce the result. Other declarative languages include Prolog (which implements logic and is used in AI) and Mathematica.

PROC SQL

The basics of the syntax for PROC SQL is (where `TABLE` refers to a SAS data set)

```
PROC SQL <options>;
SELECT Columns
FROM TABLE
WHERE Columns
GROUP BY Columns
;
QUIT;
```

```
DATA TEMP;
INPUT ID $ NAME $ SALARY DEPARTMENT $;
DATALINES;
1 Rick 623.3 IT
2 Dan 515.2 Operations
3 Michelle 611 IT
4 Ryan 729 HR
5 Gary 843.25 Finance
6 Nina 578 IT
7 Simon 632.8 Operations
8 Guru 722.5 Finance
;
RUN;

PROC SQL;
CREATE TABLE EMPLOYEES AS
SELECT * FROM TEMP;
QUIT;

PROC PRINT data = EMPLOYEES; RUN;
```

In the previous code, bot the Temp and EMPLOYEES data sets show up in the WORK library in SAS. Note that the Department value "Operations" gets truncated to 8 characters.

Obs	ID	NAME	SALARY	DEPARTMENT
1	1	Rick	623.30	IT
2	2	Dan	515.20	Operatio
3	3	Michelle	611.00	IT
4	4	Ryan	729.00	HR
5	5	Gary	843.25	Finance
6	6	Nina	578.00	IT
7	7	Simon	632.80	Operatio
8	8	Guru	722.50	Finance

PROC SQL

The previous basically used PROC SQL to just duplicate the data set created in the data step. You can also get a subset of rows using WHERE statements, and update a data set by inserting or deleting rows. You can also create new variables using mathematical operations and the "as" keyword.
Another feature is using the UPDATE statement to change a value in the data set

```
INPUT ID $ NAME $ SALARY DEPARTMENT $
DATALINES;
1 Rick 623.3 IT
2 Dan 515.2 Operations
3 Michelle 611 IT
4 Ryan 729 HR
5 Gary 843.25 Finance
6 Nina 578 IT
7 Simon 632.8 Operations
8 Guru 722.5 Finance
;
RUN;

PROC SQL;
CREATE TABLE EMPLOYEES2 AS
SELECT ID as EMPID,
Name as EMPNAME ,
SALARY as SALARY,
DEPARTMENT as DEPT,
SALARY*0.23 as COMMISION
FROM TEMP
where SALARY < 700
;
QUIT;

PROC SQL;
UPDATE EMPLOYEES2
      SET SALARY = SALARY*1.25;
   QUIT;
      PROC PRINT data = EMPLOYEES2;
RUN;
```

RESULTS OUTPUT DATA

Obs	EMPID	EMPNAME	SALARY	DEPT	COMMISION
1	1	Rick	779.125	IT	143.359
2	2	Dan	644.000	Operatio	118.496
3	3	Michelle	763.750	IT	140.530
4	6	Nina	722.500	IT	132.940
5	7	Simon	791.000	Operatio	145.544

SQL ideas

Now that we've seen a couple examples of PROC SQL, we'll discuss some of the ideas behind it.
Usually, we think of a dataset as being like a matrix with ordered observations and columns. PROC SQL is designed for multisets of data. A multiset (just like a set) has no order, but can allow repeated entries. However, for a good data base design, there is a desire to have no repeated entries.
Here is an example of working with sets versus multisets:

$$\text{Sets:} \{1,2,3\} \cup \{3,4,5\} = \{1,2,3,4,5\}$$

$$\text{Multisets} : \{1,2,3\} \cup \{3,4,5\} = \{1,2,3,3,4,5\}$$

SQL ideas

A good table for databases should also avoid columns that are largely empty (have lots of missing or NULL values). This can happen, for example, in repeated measures recorded in wide format, where most individuals are not observed at later time points.

In SQL, the term *normalization* refers to organizing data to save space (memory) and to eliminate duplication or repetition of data. (For example, instead of having duplicate rows, you could count how often each distinct row of values occurs.)

SQL: first normal form (1NF)

In first normal form, the data is in rectangular format and there is a column (such as the subject ID) that uniquely identifies each row. However, there still might be some redundant information in the table.

CUSTNUM	CUSTNAME	CUSTCITY	ITEM	UNITS	UNITCOST	MANUCITY
1	Smith	San Diego	Chair	1	$179.00	San Diego
1	Smith	San Diego	Pens	12	$0.89	Los Angeles
1	Smith	San Diego	Paper	4	$6.95	Washington
1	Smithe	San Diego	Stapler	1	$8.95	Los Angeles
7	Lafler	Spring Valley	Mouse Pad	1	$11.79	San Diego
7	Loffler	Spring Valley	Pens	24	$1.59	Los Angeles
13	Thompson	Miami	Markers	.	$0.99	Los Angeles

SQL: second normal form (2NF)

Note that in the previous example, the customer number and Smith and San Diego is repeated several times. Also, if the customer number uniquely identifies the customer, then "Smithe" must be a mispelling.

The data can be rearranged into two tables, say one with customer number, last name, and city, and the other with the remaining variables that are linked to the first table. This reduces the total number of entries.

Table 1.2: CUSTOMERS Table

CUSTNUM	CUSTNAME	CUSTCITY
1	Smith	San Diego
1	Smithe	San Diego
7	Lafler	Spring Valley
13	Thompson	Miami

Table 1.3: PURCHASES Table

CUSTNUM	ITEM	UNITS	UNITCOST	MANUCITY
1	Chair	1	$179.00	San Diego
1	Pens	12	$0.89	Los Angeles
1	Paper	4	$6.95	Washington
1	Stapler	1	$8.95	Los Angeles
7	Mouse Pad	1	$11.79	San Diego
7	Pens	24	$1.59	Los Angeles
13	Markers	.	$0.99	Los Angeles

SQL: second normal form (2NF)

Note that the original table had 7 rows and 7 columns (49 entries). The new tables are 4×3 and 7×5. The new tables have a total of $12 + 35 = 47$ entries. This is not a huge savings, but in larger data sets with more repetition could be considerable.

In a sense what has happened is that the original table is "unmerged". You could merge the two tables in 2NF form to create the table in 1NF form.

One advantage of 2NF form here is that if a customer has updated information (such as moving to a new city), that information only has to updated in one location. For the table in 1NF form, that information has to be updated in multiple locations.

SQL: third normal form (3NF)

The variable MANUCITY refers to the location of the company where the item is manufactored (or company location). This is not related to the customer number, which is considered the key column for the data. For 3NF form, each column should depend on the key. A phrase that is used is that each column should "depend on the key, the whole key, and nothing but the key". Consequently, the previous two tables violate 3NF form.

To satisfy 3NF form, you need a new key for the manufacturer and purchase number. Then the MANUCITY column will depend on this new key instead of the customer number.

Table 1.4: CUSTOMERS Table

CUSTNUM	CUSTNAME	CUSTCITY
1	Smith	San Diego
1	Smithe	San Diego
7	Lafler	Spring Valley
13	Thompson	Miami

Table 1.5: PURCHASES Table

CUSTNUM	ITEM	UNITS	UNITCOST
1	Chair	1	$179.00
1	Pens	12	$0.89
1	Paper	4	$6.95
1	Stapler	1	$8.95
7	Mouse Pad	1	$11.79
7	Pens	24	$1.59
13	Markers	.	$0.99

Table 1.6: MANUFACTURERS Table

MANUNUM	MANUCITY
101	San Diego
112	San Diego
210	Los Angeles
212	Los Angeles
213	Los Angeles
214	Los Angeles
401	Washington

SQL nromal forms

Usually 3NF form is considered good enough for database design, but there are also 4NF and 5NF forms possible in attempt to eliminate all redundant information.
Note that the 3NF form in this example actually uses more memory (has more total cellsss) than the 2NF form.

Keywords in SQL

There are several keywords in SQL that are normally not allowed to be column names, although this is not enforced in SAS SQL. (SAS SQL does not strictly follow ANSI SQL. ANSI is the American National Standards Insitute.)
The keywords are

AS	INNER	OUTER
CASE	INTERSECT	RIGHT
EXCEPT	JOIN	UNION
FROM	LEFT	UPPER
FULL	LOWER	USER
GROUP	ON	WHEN
HAVING	ORDER	WHERE

SQL ideas

Tables often have a primary key used to identify rows. I think the multiset idea here is that observations are referred to by a key rather than a row number. A foreign key can be used to link one table (data set) to another. (Think of this as something that could be used for merging the two datasets/tables).

SQL example

The book uses a database example with 6 linked tables. They are: Customers, Inventory, Invoice, Manufacturers, Products, and Purchases.

Table 1.7: Description of Columns in the Customers Table

CUSTNUM	Unique number identifying the customer.
CUSTNAME	Name of customer.
CUSTCITY	City where customer is located.

Table 1.8: Description of Columns in the Inventory Table

PRODNUM	Unique number identifying product.
MANUNUM	Unique number identifying the manufacturer.
INVENQTY	Number of units of product in stock.
ORDDATE	Date product was last ordered.
INVENCST	Cost of inventory in customer's stock room.

Table 1.9: Description of Columns in the Invoice Table

INVNUM	Unique number identifying the invoice.
MANUNUM	Unique number identifying the manufacturer.
CUSTNUM	Customer number.
PRODNUM	Product number.
INVQTY	Number of units sold.
INVPRICE	Unit price.

Table 1.10: Description of Columns in the Manufacturers Table

MANUNUM	Unique number identifying the manufacturer.
MANUNAME	Name of manufacturer.
MANUCITY	City where manufacturer is located.
MANUSTAT	State where manufacturer is located.

Table 1.11: Description of Columns in the Products Table

PRODNUM	Unique number identifying the product.
PRODNAME	Name of product.
MANUNUM	Unique number identifying the manufacturer.
PRODTYPE	Type of product.
PRODCOST	Cost of product.

Table 1.12: Description of Columns in the Purchases Table

CUSTNUM	Unique number identifying the customer.
ITEM	Name of product.
UNITS	Number of items purchased by customer.
UNITCOST	Cost of product.

PROC CONTENTS is useful for getting an overview of the different

Output 1.1: Customers CONTENTS Output

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Label
3	custcity	Char	20	Customer's Home City
2	custname	Char	25	Customer Name
1	custnum	Num	3	Customer Number

Output 1.2: Inventory CONTENTS Output

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
4	invencst	Num	6	DOLLAR10.2		Inventory Cost
2	invenqty	Num	3			Inventory Quantity
5	manunum	Num	3			Manufacturer Number
3	orddate	Num	4	MMDDYY10.	MMDDYY10.	Date Inventory Last Ordered
1	prodnum	Num	3			Product Number

Figure 1.2. Logical Database Structure

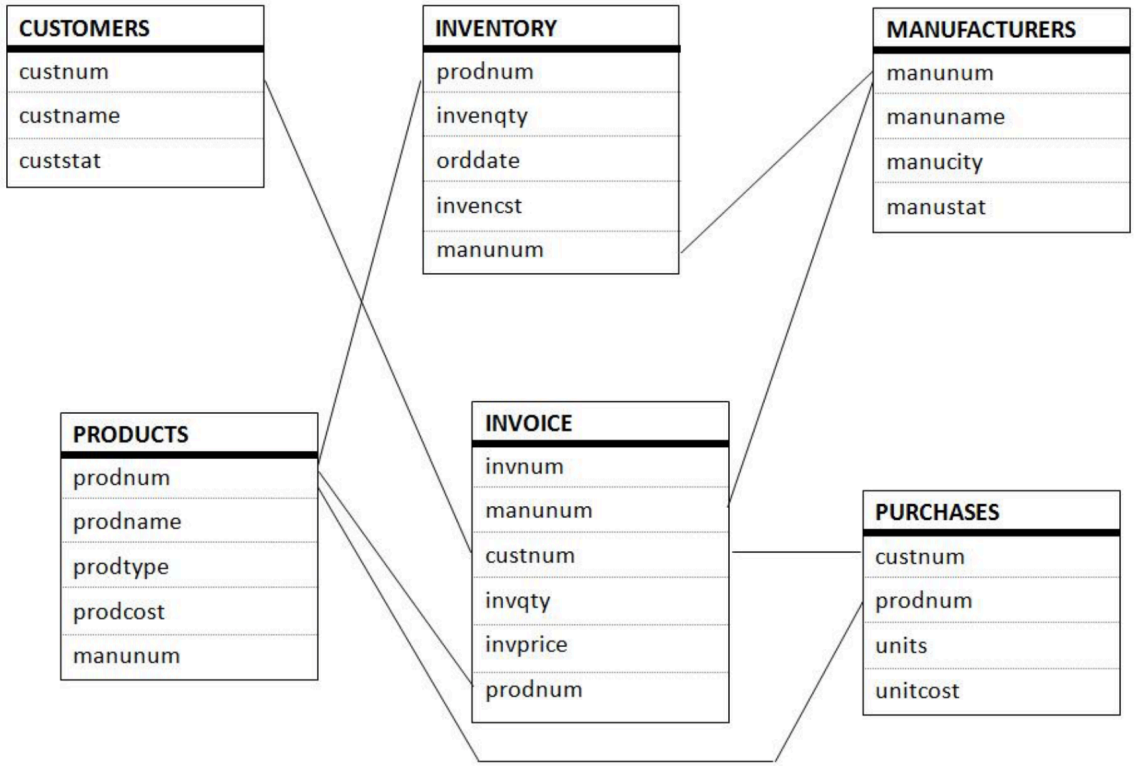


Table 1.13: CUSTOMERS Table

Obs	custnum	custname	custcity
1	101	La Mesa Computer Land	La Mesa
2	201	Vista Tech Center	Vista
3	301	Coronado Internet Zone	Coronado
4	401	La Jolla Computing	La Jolla
5	501	Alpine Technical Center	Alpine
6	601	Oceanside Computer Land	Oceanside
7	701	San Diego Byte Store	San Diego
8	801	Jamul Hardware & Software	Jamul
9	901	Del Mar Tech Center	Del Mar
10	1001	Lakeside Software Center	Lakeside
11	1101	Bonsall Network Store	Bonsall
12	1201	Rancho Santa Fe Tech	Rancho Santa Fe
13	1301	Spring Valley Byte Center	Spring Valley
14	1401	Poway Central	Poway
15	1501	Valley Center Tech Center	Valley Center
16	1601	Fairbanks Tech USA	Fairbanks Ranch
17	1701	Blossom Valley Tech	Blossom Valley
18	1801	Chula Vista Networks	
N = 18			

Table 1.14: INVENTORY Table

Obs	prodnum	invenqty	orddate	invencst	manunum
1	1110	20	09/01/2000	$45,000.00	111
2	1700	10	08/15/2000	$28,000.00	170
3	5001	5	08/15/2000	$1,000.00	500
4	5002	3	08/15/2000	$900.00	500
5	5003	10	08/15/2000	$2,000.00	500
6	5004	20	09/01/2000	$1,400.00	500
7	5001	2	09/01/2000	$1,200.00	600
N = 7					

Table 1.15: INVOICE Table

Obs	invnum	manunum	custnum	invqty	invprice	prodnum
1	1001	500	201	5	$1,495.00	5001
2	1002	600	1301	2	$1,598.00	6001
3	1003	210	101	7	$245.00	2101
4	1004	111	501	3	$9,600.00	1110
5	1005	500	801	2	$798.00	5002
6	1006	500	901	4	$396.00	6000
7	1007	500	401	7	$23,100.00	1200
N = 7						

Table 1.16: MANUFACTURERS Table

Obs	manunum	manuname	manucity	manustat
1	111	Cupid Computer	Houston	TX
2	210	Global Comm Corp	San Diego	CA
3	600	World Internet Corp	Miami	FL
4	120	Storage Devices Inc	San Mateo	CA
5	500	KPL Enterprises	San Diego	CA
6	700	San Diego PC Planet	San Diego	CA
N = 6				

Table 1.17: PRODUCTS Table

Obs	prodnum	prodname	manunum	prodtype	prodcost
1	1110	Dream Machine	111	Workstation	$3,200.00
2	1200	Business Machine	120	Workstation	$3,300.00
3	1700	Travel Laptop	170	Laptop	$3,400.00
4	2101	Analog Cell Phone	210	Phone	$35.00
5	2102	Digital Cell Phone	210	Phone	$175.00
6	2200	Office Phone	220	Phone	$130.00
7	5001	Spreadsheet Software	500	Software	$299.00
8	5002	Database Software	500	Software	$399.00
9	5003	Wordprocessor Software	500	Software	$299.00
11	5004	Graphics Software	500	Software	$299.00
N = 10					

Table 1.18: PURCHASES Table

Obs	custnum	prodnum	units	unitcost
1	1701	1110	1	$3,200.00
2	101	5001	7	$299.00
3	701	5001	11	$299.00
4	701	5003	8	$299.00
5	701	5002	4	$399.00
6	701	5004	3	$299.00
7	701	1700	2	$3,400.00
8	701	1200	3	$3,300.00
9	701	1110	2	$3,200.00
10	1301	5001	3	$299.00
11	1301	5003	5	$299.00
12	1301	5002	2	$399.00
13	901	1700	2	$3,400.00
14	901	1200	3	$3,300.00
15	901	1110	5	$3,200.00
16	901	5001	9	$299.00
17	901	5002	5	$399.00
18	901	5003	8	$299.00
19	901	5004	2	$299.00
20	401	5001	11	$299.00
21	401	5002	5	$399.00
22	401	5003	7	$299.00
23	401	5004	3	$299.00
24	401	1700	3	$3,400.00
25	401	1200	6	$3,300.00
26	201	5001	6	$299.00
27	201	5001	6	$299.00
28	201	5003	9	$299.00
29	201	5002	4	$399.00
30	201	1700	3	$3,400.00
31	901	5001	2	$299.00
32	201	5001	2	$299.00
33	201	2102	5	$175.00
34	1101	2102	9	$175.00
35	1301	2102	11	$175.00
36	1401	2102	7	$175.00
37	801	2102	5	$175.00
38	501	2102	12	$175.00
39	301	2102	8	$175.00
40	1101	2200	3	$130.00
41	101	2102	9	$175.00

Type to enter text

42	101	5003	3	$299.00
43	101	5004	2	$299.00
44	101	1200	3	$3,300.00
45	101	1700	5	$3,400.00
46	1301	1700	3	$3,400.00
47	1601	1700	7	$3,400.00
48	1801	1700	4	$3,400.00
49	1001	1700	5	$3,400.00
50	1101	1700	2	$3,400.00
51	1201	1200	8	$3,300.00
52	501	5001	3	$299.00
53	501	5003	5	$299.00
54	501	5004	1	$299.00
55	501	1700	4	$3,400.00
56	301	5001	6	$299.00
57	501	2102	9	$175.00
N = 57				

Data step versus PROC SQL

Here's an example to compare creating an empty data set with four varaibles with user defined lengths and labels.

```
DATA PURCHASES;
      LENGTH CUSTNUM  4.
             PRODNUM  3.
             UNITS    3.
             UNITCOST 4.;
      LABEL CUSTNUM  =  'Customer Number'
            PRODNUM  =  'Product Purchased'
            UNITS    =  '# Units Purchased'
            UNITCOST =  'Unit Cost';
      FORMAT UNITCOST DOLLAR12.2;
RUN;

PROC CONTENTS DATA=PURCHASES;
RUN;
```

Data step versus PROC SQL

Equivalent PROC SQL code:

```
PROC SQL;
    CREATE TABLE PURCHASES
         (CUSTNUM NUM LENGTH=4
              LABEL='Customer Number'
           PRODNUM NUM LENGTH=3
              LABEL='Product Purchased',
           UNITS NUM LENGTH=3
              LABEL='# Units Purchased'
           UNITCOST NUM LENGTH=4
              LABEL='Unit Cost');
QUIT;
```

Data step versus PROC SQL

Many step functions and formats are also available in PROC SQL, include data/time formats and string functions.

Data step versus PROC SQL

To get a list of unique values in a particular column, you can use the UNIQUE keyword in PROC SQL. How would you do this using regular data step programming?

```
PROC SQL;
   SELECT DISTINCT MANUNUM
      FROM INVENTORY;
QUIT;
```

RESULTS

Manufacturer Number
111
170
500
600

Operators in SQL

Operators available in SQL include the usual comparisons in SAS such as

- $=$ EQ
- $=$ NEQ
- $<$ LT
- $<=$ LE
- $>$ GT
- $>=$ GE

Operators in SQL

You can also compare two strings conveniently by truncating the strings to the length of the shorter string.

- EQT equal to
- GTT greater than
- LTT less than
- GET greater than or equal to
- LET less than or equal to
- NET not equal to

Operators in SQL

Logical operators can be done using AND, OR, and NOT such as `WHERE X GT 30 Y AND POSITION EQT SALES`

Arithmetic operators are the same as in SAS, such as $+$, $*$ and $**$ for exponentiation.

Concatenation in SQL

Strings can be concatenated using the usual concatenation operator in SAS, `||`, or using SELECT CAT, which works very similarly to the `paste()` function in R:

```
SELECT CAT(VAR1, "-", VAR2, "-", VAR3)
```

concatenates three columns with hyphens between them in order to make the combination a single string. This can be useful when combining two keys to make a new key.

Row numbers SQL

Although part of the philosophy of SQL is that tables are sets of rows rather than ordered matrices, you can cheat to generate a row number using the function `MONOTONIC()`. Apparently this is undocumented, and I think not an official part of SQL. Here is an example

```
PROC SQL;
  SELECT MONOTONIC() AS Row_Number FORMAT=COMMA6.,
                 PRODNUM,
                 UNITS,
                 UNITCOST
    FROM PURCHASES;
QUIT;
```

Summary stats available in SQL

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

Logic in SQL

Note that for logic programming in SQL, for loops are not supported. There are ways around this, but they are generally not recommended and not in the spirit of SQL. Conditionally processing can be done normally using WHERE and CASE statements. Here is an example, basically using WHEN...THEN instead of IF...THEN.

```
PROC SQL;
  SELECT PRODNAME,
                CASE PRODTYPE
                  WHEN 'Laptop'              THEN 'Hardware'
                  WHEN 'Phone'               THEN 'Hardware'
                  WHEN 'Software'            THEN 'Software'
                  WHEN 'Workstation'     THEN 'Hardware'
                 ELSE   'Unknown'
         END AS Product_Classification
    FROM PRODUCTS;
QUIT;
```

Logic in SQL

PROC SQL is also frequently combined with using the ODS and the macro language.