An Introduction to PROC SQL

```
PROC SQL;
  SELECT state, SUM(sales)
  FROM ussales
  WHERE state IN ('IL','WI')
  GROUP BY state;
QUIT;
```

STATE	
IL	84976.57
WI	34238.57

SYSTEMS SEMINAR CONSULTANTS, INC.

Steven First

2997 Yarmouth Greenway Drive, Madison, WI 53711

Phone: (608) 278-9964 • Web: www.sys-seminar.com

Systems Seminar Consultants, Inc.

Consulting, Training, and Support

- SAS
- SQL
- ETL
- Reporting Systems
- Business Analytics
- Data modeling
- Automating Processes
- “Big Data”

Apply good IT and systems practices to real life business applications.

Work with Marketing, Finance, Retail, Insurance, Utilities, Manufacturing, Healthcare, Government organizations.

Questions, Comments

- We have several hundred attendees, so we won't be able to answer questions live.

- Please email questions or comments to train@sys-seminar.com.

- A copy of the presentation and a recording of the webinar will be emailed out to attendees. This can be shared with people who did not attend.

Presenter – Steven J. First

- 30 years as President and Founder of SSC
- Over 30 years of SAS experience, including hundreds of manufacturing, retail, government, marketing, and financial applications.
- Founder of WISAS and WISUG
- Invited speaker at local, regional, and international SAS user groups

Why Learn PROC SQL?

Advantages of learning PROC SQL:

- SQL can retrieve information without having to learn SAS syntax
- PROC SQL requires fewer and shorter statements than traditional SAS code
- In general, it uses fewer resources than the conventional DATA and PROC steps
- Knowledge is transferable.

Today's Topics: Overview of PROC SQL

- Introduction / Features
- The SELECT Statement
- Writing reports using SQL
- Creating a SAS dataset

1 day *The SAS SQL Procedure* Web class from SSC on September 27.
Contact train@sys-seminar.com for details.

Not Covered Today...

- SET Operators
- Subqueries
- Creating Macro Variables
- SAS/ACCESS for relational data bases in depth
- Pass-thru facility of SAS/ACCESS in depth

An Introduction to PROC SQL

The features of PROC SQL:

- Base SAS® PROC
- DATA and PROC step functionalities
- similar to the ANSI standard for SQL with a few key differences
- used to retrieve, update, and report on information
- can read SAS data files
- can access other database products via SAS/ACCESS
- can build SAS datafiles and views
- in some cases is more efficient than traditional SAS code.

Notes:

- SQL is often associated with databases, ie DB2; PROC SQL is using the SQL language for SAS datasets.
- SAS datasets may be traditional SAS files or they may refer to other databases (SAS/ACCESS).
- Differences between PROC SQL and the ANSI standard SQL are listed in SAS® Guide to the SQL Procedure.

What does SQL mean?

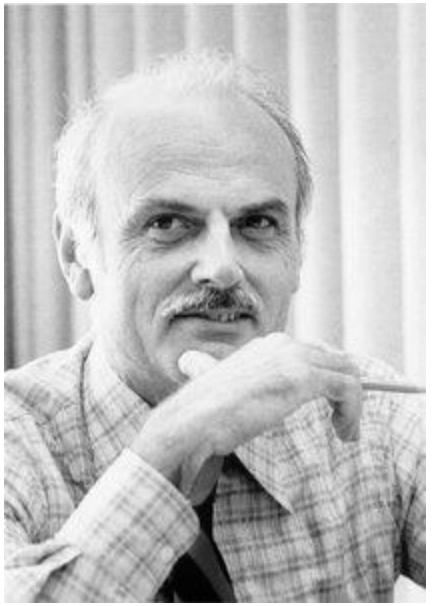
Structured Query Language

SQL is a standardized, widely used language.

SQL is often pronounced “sequel”

What is SQL?

- Origins – Authored by Dr. E.F. Codd of IBM
- ANSI Standards 1986, 1989, 1992, 1999, 2003
- DDL (data definition language) and DML (data manipulation language). We are concentrating on DML.
- Simple Syntax
  - Easy to understand data flow (multiple tables in, one table out)
  - Small number of verbs (clauses)



Terminology

The terminology in SQL is slightly different than in standard SAS, but the meaning is the same.

SAS		SQL
dataset	=	table
variable	=	column
observation	=	row

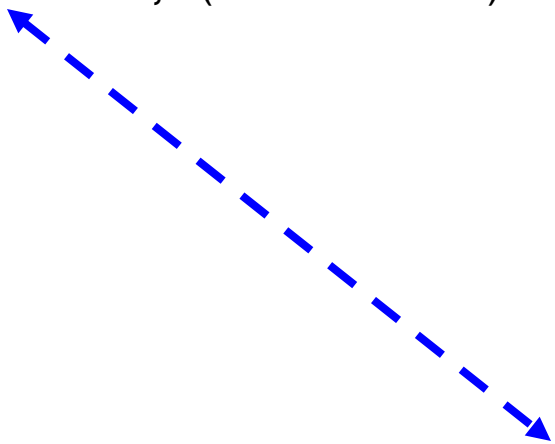
	Name	Division	Years	Sales	Expense	State
1	CHRIS	H	2	233.11	94.12	WI
2	MARK	H	5	298.12	52.65	WI
3	SARAH	S	6	301.21	65.17	MN
4	PAT	H	4	4009.21	322.12	IL
5	JOHN	H	7	678.43	150.11	WI
6	WILLIAM	H	11	3231.75	644.55	MN
7	ANDREW	S	24	1762.11	476.13	MN
8	BENJAMIN	S	3	201.11	25.21	IL

What are the features of PROC SQL?

- A base SAS Procedure
- Combines DATA and PROC step capabilities
- Similar to ANSI standard SQL syntax
- Can read SAS Data Files, Views, data bases (with SAS/ACCESS)
- Can build SAS Data Files and Views, data bases (with SAS/ACCESS)
- May be more efficient than standard SAS code

A Sample of PROC SQL Syntax

```
PROC SQL;
  SELECT STATE, SALES, (SALES * .05) AS TAX
  FROM USSALES;
QUIT;
```



Notes:

- Multiple columns are separated by **commas**
- The SELECT statement DOES NOT limit the number of columns processed (all are read in)
- At least one SELECT statement required
- The select statement names the columns and defines the order in which they will appear
- The SELECT statement can dynamically create new columns

Resulting Query (Output Window)

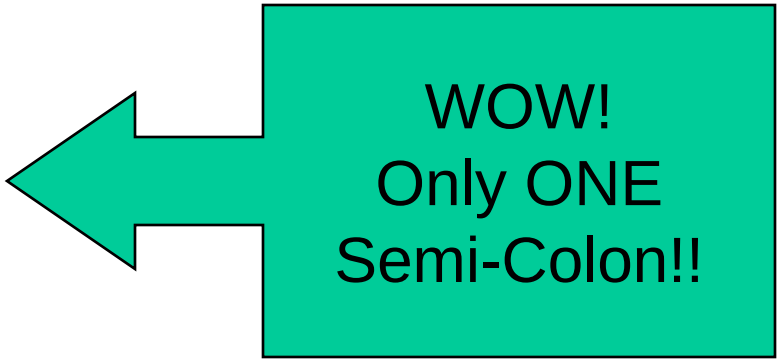
STATE	SALES	TAX
WI	10103.23	505.1615
WI	9103.23	455.1615
WI	15032.11	751.6055
MI	33209.23	1660.462
MI	20132.43	1006.622
IL	20338.12	1016.906
IL	10332.11	516.6055
IL	32083.22	1604.161
IL	22223.12	1111.156

The SELECT Statement's Syntax

```
PROC SQL options;

  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s)
   ;

QUIT;
```



Notes:

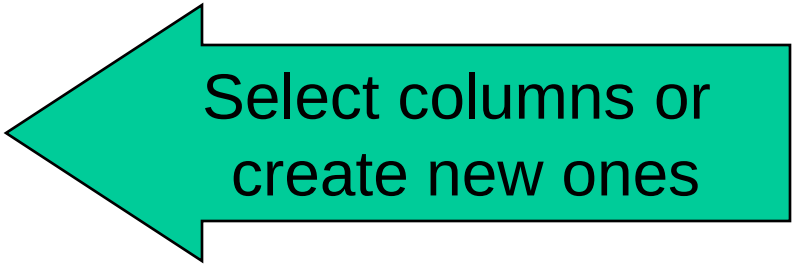
- The SELECT statement describes the appearance of the query
- It contains several clauses
- The sequence of the clauses **is important**

The SELECT Clause

```
PROC SQL options;

  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s)
   ;

QUIT;
```



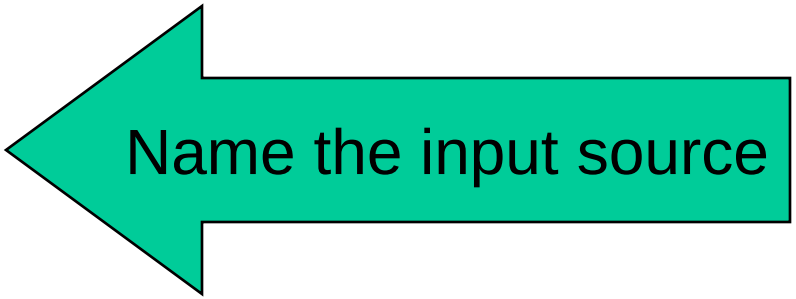
Notes:

- QUIT not required, can have more SELECT statements

The FROM Clause

```
PROC SQL options;

  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s)
   ;
```



Notes:

- The FROM table name can be a SAS data set, a view, or a DBMS table (such as Oracle or DB2)

The WHERE Clause

```
PROC SQL options;

  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s)
   ;
```

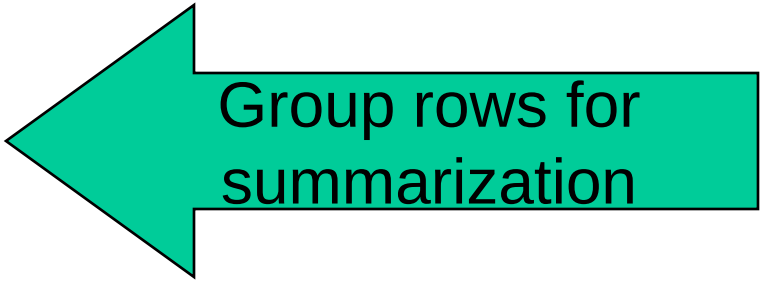
Notes:

- The WHERE clause subsets rows from the in-coming table

The GROUP BY Clause

```
PROC SQL options;

  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s)
   ;
```



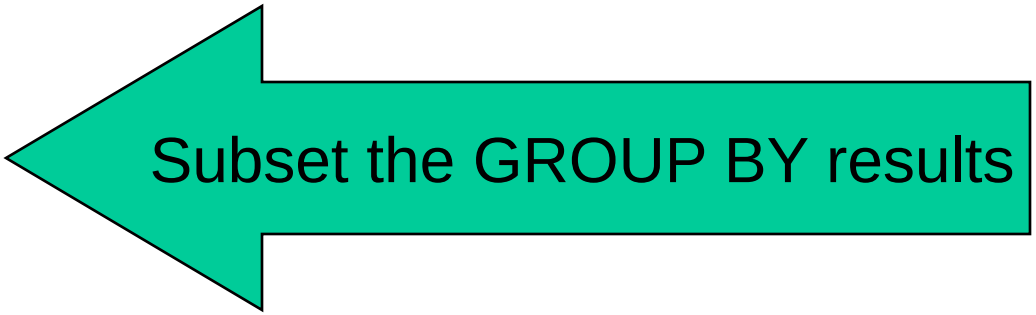
Notes:

- The GROUP BY clause specifies how to group the data for summarizing
- Similar to the CLASS statement in PROC MEANS or SUMMARY

The HAVING Clause

```
PROC SQL options;

  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s)
   ;
```



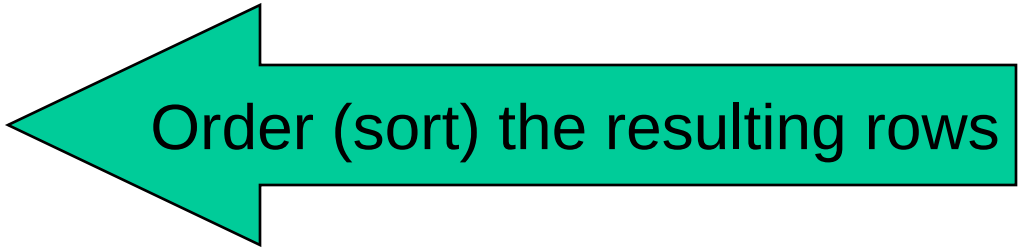
Notes:

- The HAVING clause subsets results of the GROUP BY clause (summary level)

The ORDER BY Clause

```
PROC SQL options;

  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s)
   ;
```



Notes:

- PROC SORT is NOT required, SQL will sort when doing the query

Placement of the SELECT clauses matters...

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

Acronym anyone?

SOME
FRENCH
WAITERS
GROW
HAIRY (HEALTHY?)
ORANGES

Several SELECT clauses at once

```
proc sql;
  SELECT state, sum(sales) as totsales
  FROM ussales
  WHERE state in
            ('WI','MI',’IL’)
  GROUP BY state
  HAVING  sum(sales) > 40000
  ORDER BY state desc
   ;
quit;
```

STATE	totsales
MI	53341.66
IL	84976.57

Notes:

- Column alias (i.e. column heading, new variable) defined by ‘AS’ keyword
- ‘WI’ not in report since the sum(sales) was under 40,000

Changing the Appearance of a Query Report

```
PROC SQL NUMBER DOUBLE;
  SELECT STATE, SALES
  FROM USSALES;
QUIT;
```

Row	STATE	SALES
1	WI	10103.23
2	WI	9103.23
3	WI	15032.11

Notes:

- The **NUMBER** option will print a column labeled 'ROW' containing the row number
- The **DOUBLE** option will double space the report

A Simple PROC SQL

```
OPTIONS LS=70;
PROC SQL;
  SELECT *
  FROM USSALES;
QUIT;
```

Notes:

- An asterisk on the SELECT statement selects all the columns
- Note effect of LS=70 on results

Resulting Query (Output Window)

```
STATE      SALES   STORENO
COMMENT
STORENAM
_____________________________________________________________________

WI      10103.23   32331
SALES WERE SLOW BECAUSE OF COMPETITORS SALE
RON'S VALUE RITE STORE

WI       9103.23   32320
SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER
PRICED SMART GROCERS
```

(partial results, line wraps due to LS=70)

FLOW Option

```
OPTIONS LS=70;
PROC SQL FLOW;
  SELECT *
  FROM USSALES;
QUIT;
```

Notes:

- **FLOW** allows text to wrap within the columns
- The value wraps to the length of the column if not specified

FLOW Results

STATE	SALES	STORENO	COMMENT	STORENAM
WI	10103.23	32331	SALES WERE SLOW BECAUSE OF COMPETITORS SALE	RON'S VALUE RITE STORE
WI	9103.23	32320	SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER	PRICED SMART GROCERS
WI	15032.11	32311	AVERAGE SALES ACTIVITY REPORTED	VALUE CITY
MI	33209.23	33281	SALES WERE STRONG FROM FLYER IN THE PAPER	WOODBRIDGE GROCERS
MI	20132.43	33312	PRODUCE SECTION STRONGEST SELLING	ONE STOP SHOPPING MART

(partial results; flow option keeps columns lined up)

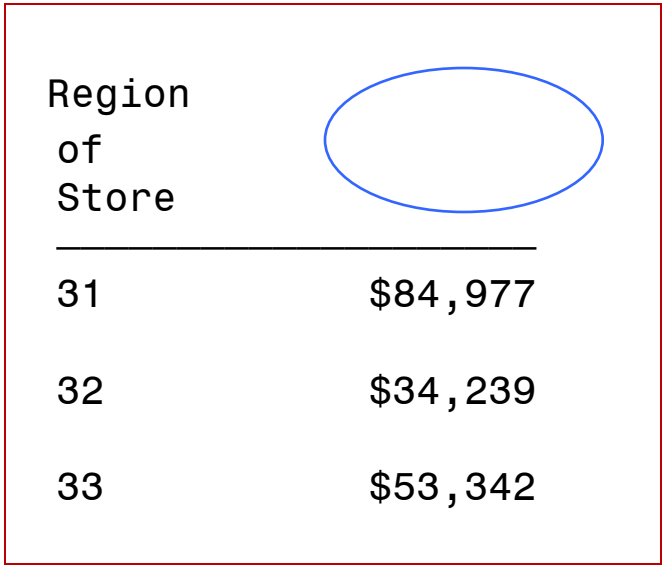
Creating New Columns

```
proc sql double;
  SELECT substr(storeno,1,2) as region   label='Region of Store',
                 sum(sales)              format=dollar12.
  FROM ussales
  GROUP BY region;
quit;
```

SAS Enhancements to ANSI Standard SQL :

- DATA step functions can be used in an expression to create a new column except LAG(), DIF(), and SOUNDEX()
- Labels, formats, and widths can be assigned as column modifiers
- Options on the Proc SQL Statement

The Resulting Output



Region of Store	
31	$84,977
32	$34,239
33	$53,342

Notes:

- Dynamic columns not given an alias or a label will appear on the report with no heading

The CALCULATED Option

```
PROC SQL INOBS=9;
  SELECT STATE, (SALES * .85) AS DISCOUNT,
    (SALES * .85) * .05 AS DISCOUNT_TAX
  FROM USSALES;

  SELECT STATE, (SALES * .85) AS DISCOUNT,
    CALCULATED DISCOUNT * .05 AS DISCOUNT_TAX
  FROM USSALES;
QUIT;
```

Same Results

Notes:

- The CALCULATED component refers to a previously calculated column - recalculation is not necessary
- The CALCULATED component must refer to a variable created in the same SELECT statement
- The CALCULATED option can be used in the SELECT and WHERE clauses.

The Resulting Output

CALCULATED column

STATE	DISCOUNT	DISCOUNT_TAX
WI	8587.746	429.3873
WI	7737.746	386.8873
WI	12777.29	638.8647
MI	28227.85	1411.392
MI	17112.57	855.6283
IL	17287.4	864.3701
IL	8782.294	439.1147
IL	27270.74	1363.537
IL	18889.65	944.4826

Enhancing the Appearance of Reports

```
TITLE 'REPORT OF THE U.S. SALES';
FOOTNOTE 'PREPARED BY THE MARKETING DEPT.';
OPTIONS LS=64 PS=16 NOCENTER;

PROC SQL;
  SELECT STATE,
         SALES FORMAT=DOLLAR10.2
               LABEL='AMOUNT OF SALES',
         (SALES * .05) AS TAX
               FORMAT=DOLLAR7.2
               LABEL='5% TAX'
  FROM USSALES;

QUIT;
```

Notes:

- Titles, Footnotes, Global Options, Formats, and Labels work like in other SAS steps

The Resulting Output

```
REPORT OF THE U.S. SALES

          AMOUNT OF
STATE        SALES    5% TAX
---------------------------
WI      $10,103.23   $505.16
WI       $9,103.23   $455.16
WI      $15,032.11   $751.61
MI      $33,209.23   1660.46

PREPARED BY THE MARKETING DEPT.
```

The CASE Expression (New Column)

```
PROC SQL;
  SELECT STATE,
    CASE
      WHEN SALES<10000 THEN 'LOW'
      WHEN SALES<15000 THEN 'AVG'
      WHEN SALES<20000 THEN 'HIGH'
       ELSE 'VERY HIGH'
    END AS SALESCAT
  FROM USSALES;
QUIT;
```

Notes:

- END is required when using the CASE
- WHENs in descending probability improve efficiency
- With no ELSE condition, missing values result

The Resulting Output

STATE	SALESCAT
WI	AVG
WI	LOW
WI	HIGH
MI	VERY HIGH
MI	VERY HIGH
IL	VERY HIGH
IL	AVG
IL	VERY HIGH
IL	VERY HIGH

Variation on the CASE

```
PROC SQL;
  SELECT STATE,
  CASE WHEN SALES <= 10000
       THEN 'LOW'
       WHEN 10001 <= SALES <= 15000
       THEN 'AVG'
       WHEN 15001 <= SALES <= 20000
       THEN 'HIGH'
       ELSE 'VERY HIGH'
    END AS SALESCAT
  FROM USSALES;
QUIT;
```

Notes:

- Output is the same as previous output

All Operators are Valid in CASE-WHEN Logic

```
PROC SQL;
  SELECT STATE,
  CASE WHEN SALES BETWEEN 0 AND 10000
       THEN 'LOW'
       WHEN SALES BETWEEN 10001 AND 15000
       THEN 'AVG'
       WHEN SALES BETWEEN 15001 AND 20000
       THEN 'HIGH'
       ELSE 'VERY HIGH'
    END AS SALESCAT
  FROM USSALES;
QUIT;
```

Available operators in CASE-WHEN and WHERE Logic:

- All operators that IF uses (= , <, >, NOT, NE, AND, OR, IN, etc)
- BETWEEN AND
- CONTAINS or '?'
- IS NULL or IS MISSING
- = *
- LIKE

GROUP BY Summarization

```
PROC SQL;
  SELECT STATE, SUM(SALES) AS TOTSALES
  FROM USSALES
  GROUP BY STATE;
QUIT;
```

```
STATE  TOTSALES
 ---------------
 IL      84976.57
 MI      53341.66
 WI      34238.57
```

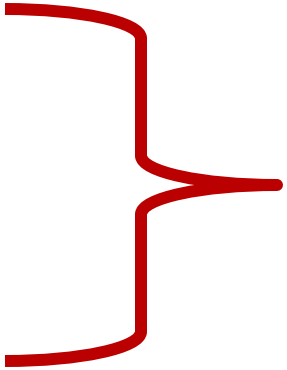
Notes:

- GROUP BY summarizes
- Use summary functions on the numeric columns for statistics
- Other summary functions: AVG/MEAN, MAX, MIN, COUNT/FREQ/N, NMISS, STD, SUM, and VAR

Summary Use without GROUP BY

Remerging will occur when a summary function is used without a GROUP BY (grand total on EVERY line)

```
PROC SQL;
  SELECT STATE,
    SUM(SALES) AS TOTSALES
  FROM USSALES;
QUIT;
```



STATE	TOTSALES
WI	172556.8
WI	172556.8
WI	172556.8
MI	172556.8
MI	172556.8
IL	172556.8
IL	172556.8

From the SAS Log:

```
NOTE: The query requires remerging summary
statistics back with the original data.
```

Remerging without GROUP BY

```
PROC SQL;
  SELECT SUM(SALES) AS TOTSALES
  FROM USSALES;
```

```
TOTSALES
--------
172556.8
```

```
  SELECT STATE, SALES,
    (SALES/SUM(SALES)) AS PCTSALES
    FORMAT=PERCENT7.2
  FROM USSALES;
QUIT;
```

```
STATE      SALES   PCTSALES
-------------------------
WI      10103.23      5.86%
WI       9103.23      5.28%
WI      15032.11      8.71%
MI      33209.23      19.2%
```

Notes:

- Remerging allows you to calculate percentages easily
- The LOG warns you when remerging occurs

Sorting Data

```
PROC SQL;
  SELECT STATE, SALES
  FROM USSALES
  ORDER BY STATE, SALES DESC;
QUIT;
```

STATE	SALES
IL	32083.22
IL	22223.12
IL	20338.12
IL	10332.11
MI	33209.23

Notes:

- Data in sorted order will not resort
- Use the DESC option for descending order
- PROC SQL and PROC SORT are nearly comparable in terms of efficiency when sorting existing columns

A Variation on the ORDER BY

```
PROC SQL;
  SELECT SUBSTR(STORENO,1,3)
    LABEL='REGION',
    (SALES * .05) AS TAX
  FROM USSALES
  ORDER BY 1 ASC, TAX DESC;
QUIT;
```

Notes:

- SQL may be more efficient when sorting on dynamically created columns (PROC SORT cannot do this!!)

- Columns can be referred to by position or name on the ORDER BY

Subsetting Using the WHERE Clause

Select only specified rows for the output.

Character

```
SELECT *
FROM USSALES
WHERE STATE IN
  ('OH','IN','IL');
```

Numeric

```
SELECT *
  FROM USSALES
  WHERE NSTATE IN (10, 20 ,30);
```

Compound

```
SELECT *
  FROM USSALES
  WHERE STATE IN
    ('OH','IN','IL')
    AND SALES > 500;
```

Be Careful of the WHERE

```
PROC SQL;
  SELECT STATE, SALES,
    (SALES * .05) AS TAX
  FROM USSALES
  WHERE STATE IN
    ('OH','IN','IL')
    AND TAX > 10 ;
QUIT;
```

Notes:

- WHERE clause cannot reference a computed column

```
141  PROC SQL;
142    SELECT STATE, SALES,
143      (SALES * .05) AS TAX
144    FROM USSALES
145    WHERE STATE IN
146      ('OH','IN','IL')
147      AND TAX > 10 ;
ERROR: The following columns were not found in the contributing
       tables: TAX.
```

Recompute on the WHERE Clause

```
PROC SQL;
  SELECT STATE, SALES,
    (SALES * .05) AS TAX
  FROM USSALES
  WHERE STATE IN
    ('OH','IL','IN')
    AND (SALES * .05) > 10;
QUIT;
```

```
STATE      SALES        TAX
-------------------------
WI      10103.23   505.1615
WI       9103.23   455.1615
WI      15032.11   751.6055
IL      20338.12   1016.906
```

Notes:

- Calculated keyword could have also been used.

WHERE with GROUP BY

```
PROC SQL;
  SELECT STATE, STORENO,
    SUM(SALES) AS TOTSALES
  FROM USSALES
  GROUP BY STATE, STORENO
  WHERE TOTSALES > 500;
QUIT;
```

Notes:

- WHERE cannot be used with summary variables when using the GROUP BY. (see next slide for resulting log)

The Resulting Log

```
94   PROC SQL;
95            SELECT STATE, STORENO,SUM(SALES) AS TOTSALES
96            FROM USSALES
97             GROUP BY STATE
98             WHERE TOTSALES > 500;
               -----
               22
               202
ERROR 22-322: Expecting one of the following: (, **, *, /,
+, -, !!, ||, <, <=, <>, =, >,
               >=, EQ, GE, GT, LE, LT, NE, ^=, ~=, &, AND, !, OR,
|, ',', HAVING, ORDER.
               The statement is being ignored.

ERROR 202-322: The option or parameter is not recognized.

99   QUIT;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: The PROCEDURE SQL used 0.05 seconds.
```

Fix by Using the HAVING Clause

```
PROC SQL;
  SELECT STATE, STORENO,
    SUM(SALES) AS TOTSALES
  FROM USSALES
  GROUP BY STATE, STORENO
  HAVING SUM(SALES) > 500;
QUIT;
```

STATE	STORENO	TOTSALES
IL	31212	10332.11
IL	31373	22223.12
IL	31381	32083.22
IL	31983	20338.12
MI	33281	33209.23

Notes:

- To subset data when grouping is in effect, HAVING must be used

Checking for Duplicates

```
PROC SQL;
  SELECT CUSTID
  FROM CONTACTS
  GROUP BY CUSTID
  HAVING COUNT(*) > 1;
QUIT;
```

Duplicate Customers

CUSTID
10006
10010
10015
10017
10021

Notes:

- Summary function does not need to be on the select statement.

Creating Tables

```
PROC SQL;
  CREATE TABLE SUMSALE AS
  SELECT STATE,
         SUM(SALES) AS TOTSALES
  FROM USSALES
  GROUP BY STATE;
QUIT;

PROC PRINT DATA=SUMSALE;
RUN;
```

Obs	STATE	TOTSALES
1	IL	84976.57
2	MI	53341.66
3	WI	34238.57

Notes:

• When a CREATE statement is used in conjunction with a SELECT statement, a report will not be generated.

Creating Tables - SAS LOG

```
118  PROC SQL;
119    CREATE TABLE SUMSALE AS
120    SELECT STATE,
121           SUM(SALES) AS TOTSALES
122    FROM USSALES
123    GROUP BY STATE;
NOTE: Table WORK.SUMSALE created, with 3 rows and 2 columns.

124  QUIT;
NOTE: PROCEDURE SQL used:
      real time           0.13 seconds
      cpu time            0.00 seconds

125
126  PROC PRINT DATA=SUMSALE;
127  RUN;

NOTE: There were 3 observations read from the data set WORK.SUMSALE.
NOTE: PROCEDURE PRINT used:
      real time           0.10 seconds
      cpu time            0.00 seconds
```

Joining Data In SQL

Some of the different types of joins in PROC SQL:

- Cartesian Join
- Inner Join
- Outer Join
  - Left Join
  - Right Join
  - Full Join

Notes:

- Data need not be pre-sorted before joining
- Up to 32 tables can be joined in one query (16 pre-v8)

A Cartesian Join

- Combine all rows from one file with all rows from another file.

TABLE DATA1

VAR1	VAR2
ABC	10
GHI	15
MNO	20

TABLE DATA2

VAR1	VAR3
DEF	20
JKL	25
PQR	30

```
PROC SQL;
 SELECT *
 FROM DATA1, DATA2;
QUIT;
```

Results

VAR1	VAR2	VAR1	VAR3
ABC	10	DEF	20
ABC	10	JKL	25
ABC	10	PQR	30
GHI	15	DEF	20
GHI	15	JKL	25
GHI	15	PQR	30
MNO	20	DEF	20
MNO	20	JKL	25
MNO	20	PQR	30

A Cartesian Join - Dating Service ??

- Combine all rows from one file with all rows from another file.

TABLE GIRLS

NAME	STATE
NANCY	WI
JEAN	MN
AMELIA	IL

TABLE BOYS

NAME	STATE
NED	WI
GENE	NY
ADAM	CA

```
PROC SQL;
 SELECT *
 FROM GIRLS, BOYS;
QUIT;
```

Results

NAME	STATE	NAME	STATE
NANCY	WI	NED	WI
NANCY	WI	GENE	NY
NANCY	WI	ADAM	CA
JEAN	MN	NED	WI
JEAN	MN	GENE	NY
JEAN	MN	ADAM	CA
AMELIA	IL	NED	WI
AMELIA	IL	GENE	NY
AMELIA	IL	ADAM	CA

An Inner/Conventional Join

We want the observation only if it is in both data sets.

TABLE GIRLS

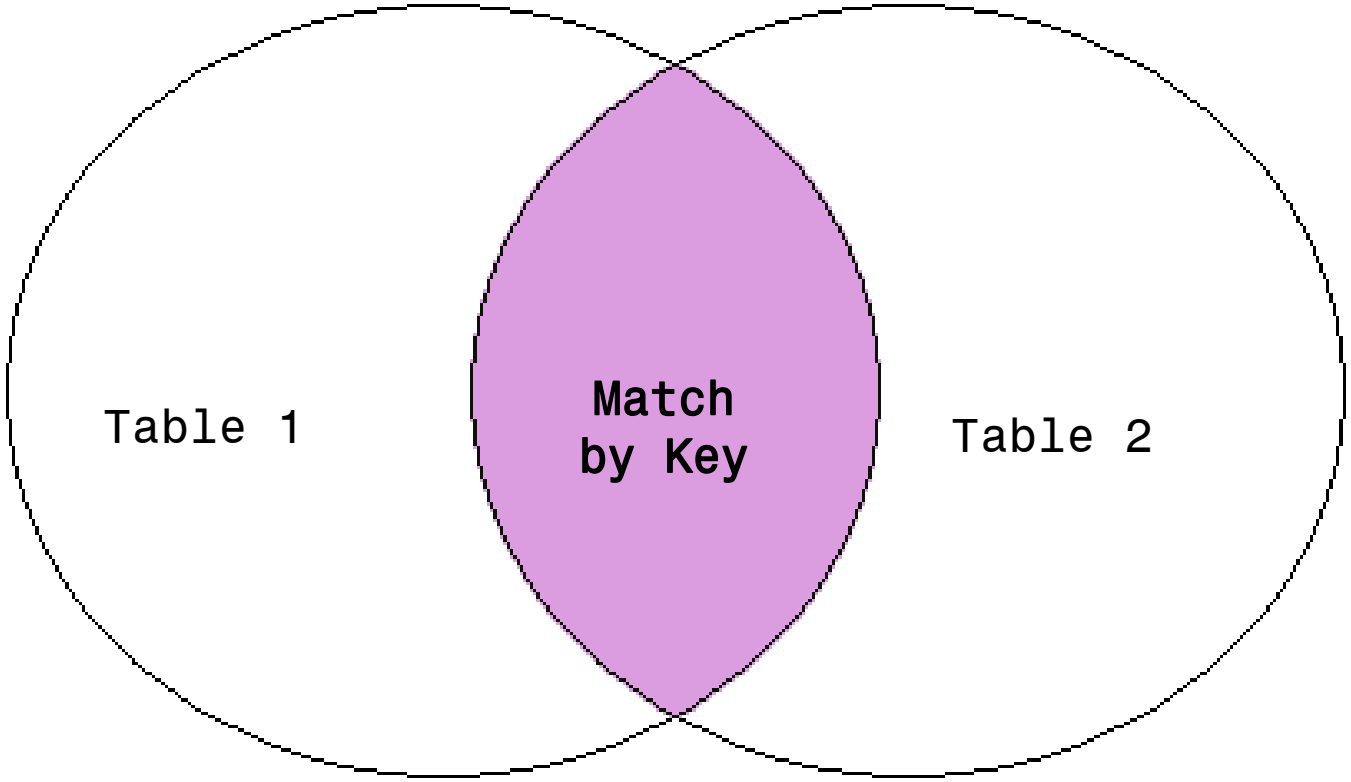
NAME	STATE
NANCY	WI
JEAN	MN
AMELIA	IL

TABLE BOYS

NAME	STATE
NED	WI
GENE	NY
ADAM	CA

```
PROC SQL;
   SELECT *
   FROM GIRLS, BOYS
   WHERE GIRLS.STATE=BOYS.STATE;
QUIT;
```

An Inner/Conventional Join



Inner Joins

```
(Results of prior example)

   NAME       STATE      NAME       STATE
   ---------------------------------------
   NANCY      WI         NED        WI
```

This could be a match-merge in a traditional SAS code:

```
DATA MERGEIT;
  MERGE GIRLS(IN=ONG) BOYS(IN=ONB);
  BY STATE;
  IF ONG AND ONB;
QUIT;
```

Notes:

- By default, STATE will appear twice in SQL report, not in the Data Step

A One-To-Many Merge In SQL

Multiple occurrences of an observation in one of the data sets.

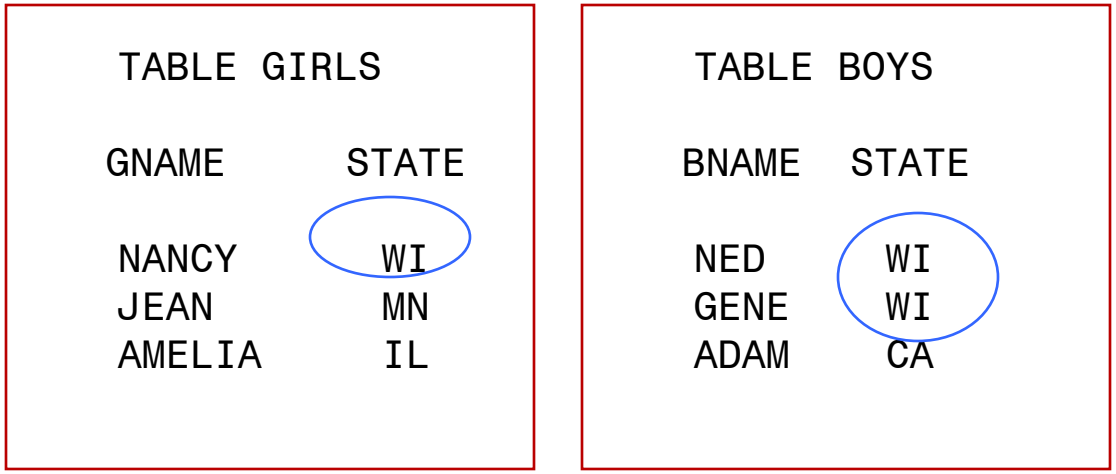


TABLE GIRLS

GNAME	STATE
NANCY	WI
JEAN	MN
AMELIA	IL

TABLE BOYS

BNAME	STATE
NED	WI
GENE	WI
ADAM	CA

Alias is used to designate data sets - eliminates dup STATE

```
PROC SQL;
   SELECT G.STATE, GNAME, BNAME
   FROM GIRLS G, BOYS B
   WHERE G.STATE=B.STATE;
QUIT;
```

The Resulting Output

STATE	GNAME	BNAME
WI	NANCY	NED
WI	NANCY	GENE

This Data Step merge result is the same as the prior SQL:

```
DATA MERGEIT;
   MERGE GIRLS(IN=ONG) BOYS(IN=ONB);
   BY STATE;
   IF ONG AND ONB;
RUN;
```

Many to Many Merge in SQL

Multiple occurrences of an observation on both data sets

TABLE GIRLS

GNAME	STATE
NANCY	WI
JEAN	WI
AMELIA	IL

TABLE BOYS

BNAME	STATE
NED	WI
GENE	WI
ADAM	CA

```
PROC SQL;
   SELECT *
   FROM GIRLS G, BOYS B
   WHERE G.STATE=B.STATE;
QUIT;
```

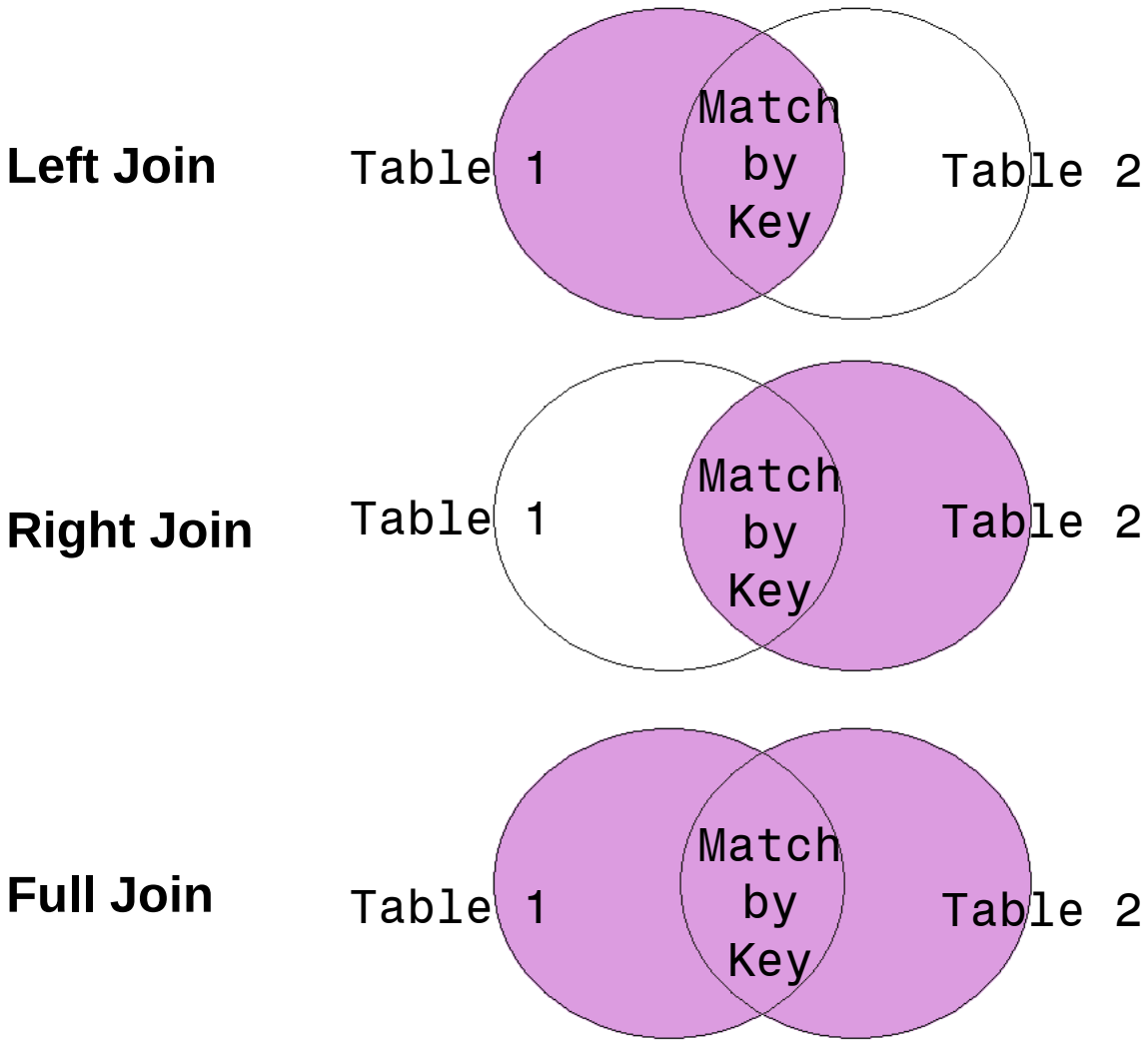
The Resulting Output

GNAME	STATE	BNAME	STATE
NANCY	WI	NED	WI
NANCY	WI	GENE	WI
JEAN	WI	NED	WI
JEAN	WI	GENE	WI

Notes:

- Many to Many Merge produces a warning message in Data Step Merge
- BEWARE - SQL does not produce a warning message

Outer Joins in SQL



Outer Joins in SQL

LEFT, RIGHT or FULL Joins

```
PROC SQL;
 SELECT *
 FROM GIRLS A
      FULL JOIN
      BOYS B
 ON A.STATE=B.STATE;
QUIT;
```

GNAME	STATE	BNAME	STATE
		ADAM	CA
AMELIA	IL		
NANCY	WI	NED	WI
NANCY	WI	GENE	WI
JEAN	WI	NED	WI
JEAN	WI	GENE	WI

Notes:

- **Left** Join keeps all records from left side of From (If ONA)
- **Right** Join keeps all records from right side of From (If ONB)
- **Full** join keeps records from both data sets

Joining More than Two Tables

The Problem:
We want a report with all claims over $1000.
It must contain the first & last name, the claim amount, the store number, & the state.

BENEFITS

FNAME	LNAME	CLAIMS
ANN	BECKER	2003
CHRIS	DOBSON	100
ALLEN	PARK	10392
BETTY	JOHNSON	3832

EMPLOYEE

FNAME	LNAME	STORENO
ANN	BECKER	33281
CHRIS	DOBSON	33281
EARL	FISHER	33281
ALLEN	PARK	31373
BETTY	JOHNSON	31373
KAREN	ADAMS	31373

FEBSALES

STATE	SALES	STORENO
WI	9103.23	32320
WI	8103.23	32331
WI	10103.23	32320

Joining More Than Two Tables (continued)

Inner Join:

```
PROC SQL;

  SELECT B.FNAME, B.LNAME,
       CLAIMS, E.STORENO, STATE

  FROM BENEFITS B,
       EMPLOYEE E,
       FEBSALES F
  WHERE B.FNAME=E.FNAME AND B.LNAME=E.LNAME
      AND E.STORENO=F.STORENO AND
      CLAIMS > 1000;

QUIT;
```

The Output:

FNAME	LNAME	CLAIMS	STORENO	STATE
ANN	BECKER	2003	33281	MI
ALLEN	PARK	10392	31373	IL
BETTY	JOHNSON	3832	31373	IL

Inner Join in Data Steps

Requires **multiple** data passes and sorts

```
PROC SORT DATA=BENEFIT;
  BY FNAME LNAME;
RUN;

PROC SORT DATA=EMPLOYEE;
  BY FNAME LNAME;
RUN;

DATA TEST1;
  MERGE BENEFIT(IN=ONA) EMPLOYEE(IN=ONB);
  BY FNAME LNAME;
  IF ONA AND ONB;
  IF CLAIMS > 1000;
RUN;
```

Inner Join in Data Steps (continued)

```
PROC SORT DATA=TEST1;
  BY STORENO;
RUN;

PROC SORT DATA=FEBSALES;
  BY STORENO;
RUN;

DATA TEST2;
  MERGE TEST1(IN=ONA) FEBSALES(IN=ONB);
  BY STORENO;
  IF ONA AND ONB;
RUN;

PROC PRINT DATA=TEST2;
RUN;
```

A DB2 PROC SQL Pass-Through Example

Produce a report of all fields and 5 rows.

```
title 'DB2 Pass Through Query';
proc sql outobs=5;
connect to db2(ssid=ssc1);
  select *                    /* sas select           */
  from connection to db2
(select   *                   /* start pass-thru query*/
   from ssctrain.benefits     /* authid and table     */
 );                           /* end  pass-thru query */
%put &sysdbrc &sysdbmsg;      /* return codes         */
disconnect from db2;
quit;
```

Notes:

- Everything inside the parentheses refers to DB2 names, values. (BLUE)
- Everything outside the parentheses is SAS. (RED)

In Conclusion

PROC SQL is a powerful data analysis tool:

- It can perform many of the same operations as found in traditional SAS code, but can often do it more efficiently because of its dense language structure.
- PROC SQL is an effective tool for joining data, particularly when doing associative, or three-way joins
- Allows use of common SQL syntax
- For more information see the documentation for PROC SQL

Next Steps

The SAS SQL Procedure

September 27, 2012, 1 Day Live Web Training, $550/Student

- an introduction to PROC SQL
- additional SELECT clauses
- the CREATE statement
- queries with set operators
- subqueries in SQL
- joining data with SQL
- creating and modifying data

Email train@sys-seminar.com for details or registration.

Free Advanced SQL Webinars

Coming This Fall – Topics and Date to Be Announced

Upcoming Lunch and Learns

Principles of System Design – August 16, 2012

Teresa Schudrowitz

SAS Efficiencies – September 13, 2012

Thomas Miron

I Have Enterprise Guide: Now What? – October 11, 2012

Jennifer First

For registration, email train@sys-seminar.com.

Contact Us

SYSTEMS SEMINAR CONSULTANTS, INC.

SAS® Training, Consulting, & Help Desk Services

2997 Yarmouth Greenway Drive • Madison, WI 53711
(608) 278-9964 • www.sys-seminar.com

SAS Alliance Silver Member

Steven First
President
sfirst@sys-seminar.com

train@sys-seminar.com