

Advanced Subqueries In PROC SQL



```
PROC SQL;  
  SELECT STATE,  
         AVG(SALES) AS AVGSALES  
  FROM  USSALES  
  GROUP BY STATE  
  HAVING AVG(SALES) >  
         (SELECT AVG(SALES)  
          FROM USSALES);  
QUIT;
```

STATE	AVGSALES
IL	21244.14
MI	26670.83



SYSTEMS SEMINAR CONSULTANTS, INC.

Steve First

2997 Yarmouth Greenway Drive, Madison, WI 53711

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

Advanced Subqueries In PROC SQL



This paper was written by Systems Seminar Consultants, Inc. SSC specializes in SAS software and offers:

- SAS Training Services
- Consulting Services
- SAS Support Plans
- Newsletter Subscriptions to *The Missing Semicolon*[™]

COPYRIGHT© 2009 Systems Seminar Consultants, Inc.

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without prior written permission of SSC. SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. *The Missing Semicolon* is a trademark of Systems Seminar Consultants, Inc.

Review of PROC SQL Basics



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

Terminology



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

<u>SAS</u>		<u>SQL</u>
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 Does SQL Mean?



Structured Query Language

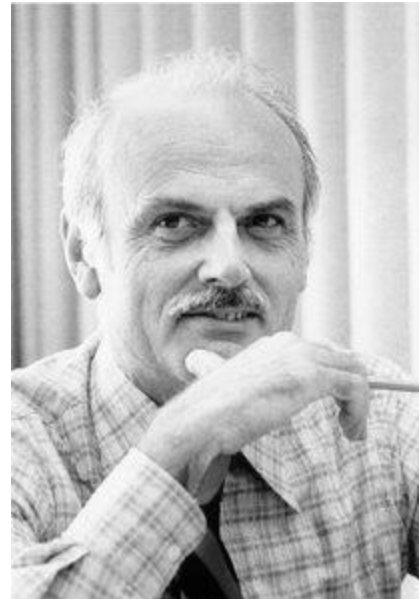
SQL is a standardized, widely used language.

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)



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)

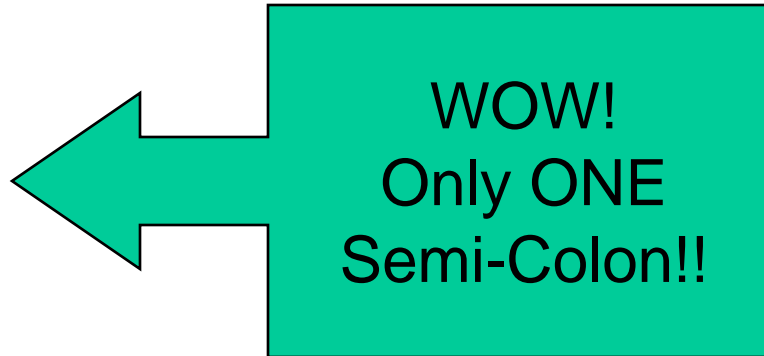


STATE	SALES	TAX
WI	10103.23	505.1615
WI	9103.23	455.1615
WI	15032.11	751.6055
MI	33209.23	1660.462
MI	20132.43	1006.622
IL	20338.12	1016.906
IL	10332.11	516.6055
IL	32083.22	1604.161
IL	22223.12	1111.156

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;
```



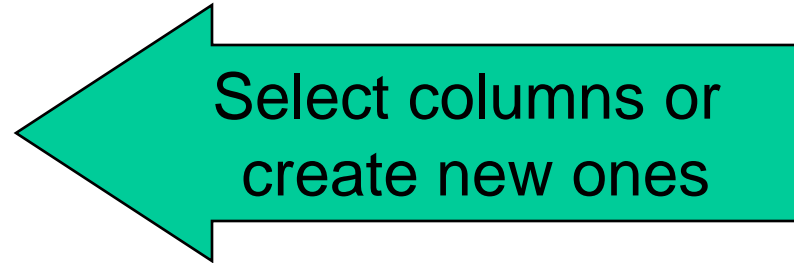
Notes:

- 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;
```



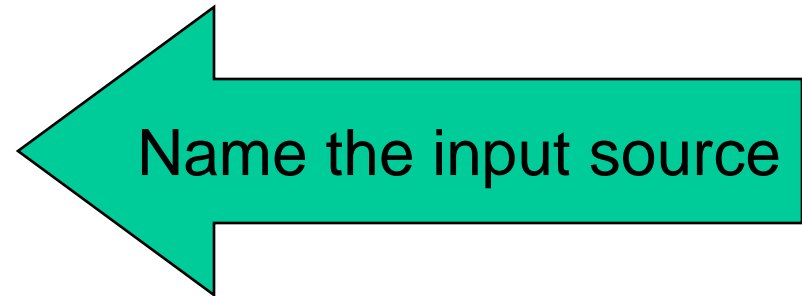
Notes:

- 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)  
  ;
```



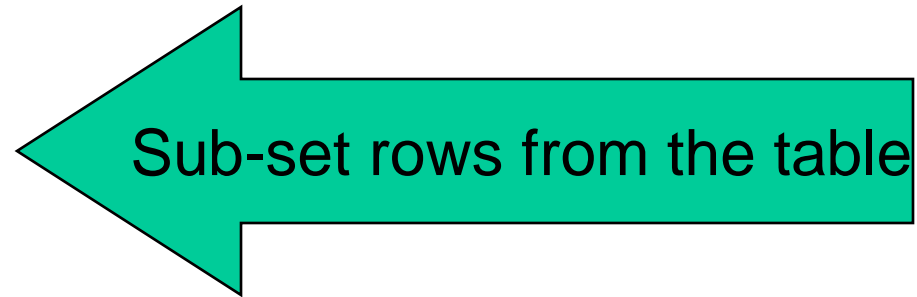
Notes:

- 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)  
  ;
```



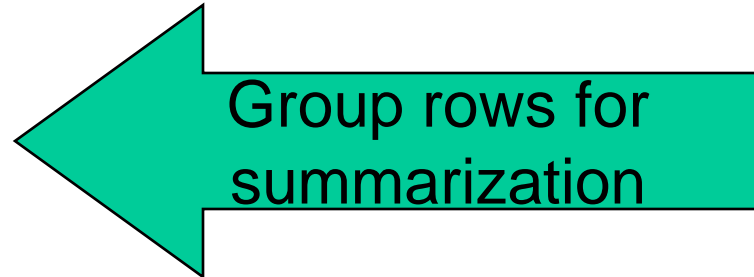
Notes:

- 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)  
  ;
```



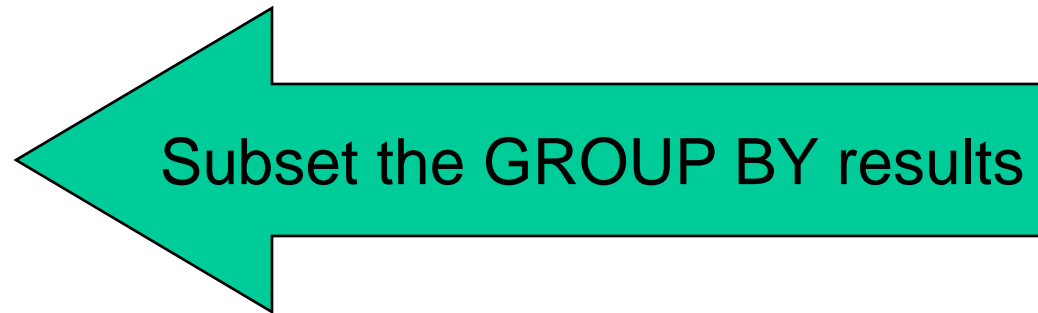
Notes:

- 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)  
  ;
```



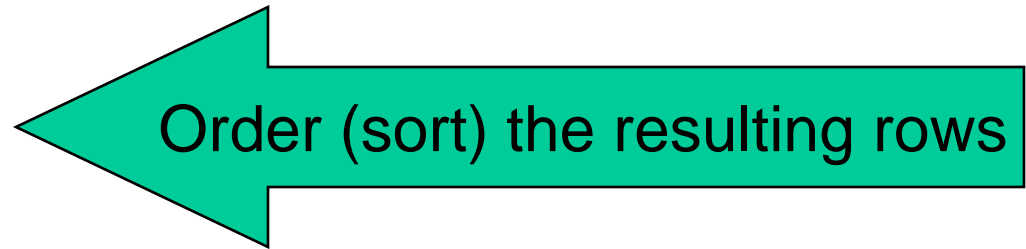
Notes:

- 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...



SSELECT
FFROM
WWHERE
GGROUP BY
HHAVING
OORDER BY

**Acronym
anyone?**

SSOME
FFRENCH
WWAITERS
GGROW
HHAIRY (HEALTHY?)
OORANGES

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

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

SELECT Clause – INTO – Create Macro Variables



The SELECT clause can also be used to:

- Create Macro Variables

Example:

- * USE PROC SQL TO BUILD MACRO VARIABLE;
- * THE 'INTO :MACRO-VARIABLE-NAME' BUILDS THE;
- * MACRO VARIABLE FROM RETURNED ROWS;

```
PROC SQL;  
  SELECT CODE  
    INTO :MINCODES SEPARATED BY ','  
      FROM CODES;  
RUN;  
  
%PUT MACRO VARIABLE 'MINCODES' = &MINCODES;
```

SAS LOG:

```
335 %PUT MACRO VARIABLE 'MINCODES' = &MINCODES;  
SYMBOLGEN: Macro variable MINCODES resolves to 123,456,789  
MACRO VARIABLE 'MINCODES' = 123,456,789
```

SELECT Clause – INTO – Use Macro Variable



Use the Macro Variable in a WHERE statement.

Example:

```
* COULD USE IN PROC PRINT;  
PROC PRINT DATA=NAMES;  
  WHERE NUMBER IN (&MINCODES);  
RUN;
```

Obs	NUMBER	NAME
1	123	DAVE
4	456	MARY
7	789	LINDA

```
* COULD ALSO USE IN SQL QUERY;  
PROC SQL;  
  SELECT *  
  FROM NAMES  
  WHERE NUMBER IN (&MINCODES);  
QUIT;
```

NUMBER	NAME
123	DAVE
456	MARY
789	LINDA

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

STATE	AMOUNT OF 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

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

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;
```

WHERE with GROUP BY (error)



```
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;
```

Notes:

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

Duplicate Customers	
	CUSTID
	10006
	10010
	10015
	10017
	10021

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)

What is a Subquery?



A subquery (inner query) is a query-expression that is nested as part of another query-expression.

- Subqueries are coded within parentheses.
- Results of the subquery are to be used as value(s) within the outer select.
- Subqueries, also known as inner queries, are evaluated before the outer query.
- Subqueries can reference the same data set as the outer query.
- Depending on the clause that contains it, a subquery can return a single value or multiple values.
- Subqueries are usually used with WHERE and HAVING expressions.
- Subqueries can also be used as part of the FROM and SELECT expressions
- Subqueries can be nested several levels deep.

Single-Value Subqueries



- Returns a single row and column.
- Can be used in a WHERE or HAVING clause with a comparison operator.
- It must return only one value or the query fails.

Which states have average sales greater than the company's average sales?

```
PROC SQL;  
    SELECT STATE, AVG(SALES) AS AVGSALES  
    FROM USSALES  
    GROUP BY STATE  
    HAVING AVG(SALES) >  
        (SELECT AVG(SALES)  
         FROM USSALES) ;  
QUIT;
```

Single-Value Subqueries



The subquery is evaluated first and returns the overall average (19172.98) to the outer query.

The effective outer query is:

```
PROC SQL;  
    SELECT STATE, AVG(SALES) AS AVGSALES  
    FROM USSALES  
    GROUP BY STATE  
    HAVING AVG(SALES) > 19172.98  
;  
QUIT;
```

STATE	AVGSALES
IL	21244.14
MI	26670.83

Multiple-Value Subqueries



- Returns more than one value from one column.
- Are used in HAVING or WHERE expression that contains IN operator or that is modified by ANY or ALL.

Evaluating More Than One Row (Error)



More than one row cannot be returned without additional options.

```
PROC SQL;  
    SELECT STATE, STORENO, SALES  
    FROM FEBSALES  
    WHERE STATE IN ('WI', 'IL')  
           AND SALES < (SELECT SALES  
                        FROM JANSALES) ;  
QUIT;
```

The resulting log (partial):

```
550 PROC SQL;  
551     SELECT STATE, STORENO, SALES  
552     FROM FEBSALES  
553     WHERE STATE IN ('WI', 'IL') AND SALES <  
554         (SELECT SALES  
555         FROM JANSALES);  
ERROR: Subquery evaluated to more than one row.  
NOTE: The SAS System stopped processing this step because of  
errors.
```

The ALL Keyword Subquery Option



The comparison is true for *all* values returned on the subquery.

```
PROC SQL ;
  SELECT STATE, STORENO, SALES
  FROM FEBSALES
  WHERE STATE IN ('WI', 'IL') AND SALES < ALL
    (SELECT SALES
     FROM JANSALES) ;
QUIT ;
```

The resulting output:

STATE	STORENO	SALES
WI	32331	8103.23
IL	31212	8332.11

Notes;

- This selects rows where SALES from FEBSALES is less than ALL the values from JANSALES or, in effect, less than the minimum value found in JANSALES.

Another Way to Select Rows Less Than Minimum



A subquery using MIN returns the same results.

```
PROC SQL;  
    SELECT STATE, STORENO, SALES  
    FROM FEBSALES  
    WHERE STATE IN ('WI', 'IL') AND SALES <  
           (SELECT MIN(SALES) FROM JANSALES) ;  
QUIT;
```

The resulting output:

STATE	STORENO	SALES
WI	32331	8103.23
IL	31212	8332.11

The ANY Keyword Subquery Option



The comparison is true for **any** one of the values returned on the subquery.

```
PROC SQL ;  
  SELECT STATE, SALES  
  FROM FEBSALES  
  WHERE STATE IN ('WI', 'IL') AND SALES < ANY  
    (SELECT SALES  
     FROM JANSALES) ;
```

```
QUIT ;
```

The resulting output:

STATE	SALES
-----	-----
WI	9103.23
WI	8103.23
WI	10103.23
WI	13032.11
IL	25338.12
IL	8332.11
IL	30083.22
IL	26223.12

The ANY Keyword Subquery Option (continued)



Notes:

- This selects rows where sales from FEBSALES is less than ANY of the JANSALES values or, in effect, less than the maximum value found on JANSALES.

Another Way to Select Less Than Maximum



The MAX function acts like the ANY option..

```
PROC SQL;  
  SELECT STATE, SALES  
  FROM FEBSALES  
  WHERE STATE IN ('WI', 'IL') AND SALES <  
    (SELECT MAX(SALES)  
     FROM JANSALES) ;  
QUIT;
```

The resulting output:

STATE	SALES
WI	9103.23
WI	8103.23
WI	10103.23
WI	13032.11
IL	25338.12
IL	8332.11
IL	30083.22
IL	26223.12

The IN Condition



- IN compares each outer row to the list of values returned by the subquery.
- COMPRESS and concatenation can be used to construct a unique key.

Example:

Who are the employees and which stores do they work for that have had an insurance claim?

```
PROC SQL;  
  SELECT FNAME, LNAME, STORENO  
  FROM EMPLOYEE  
  WHERE COMPRESS(FNAME !! LNAME) IN  
    (SELECT COMPRESS(FNAME !! LNAME)  
     FROM BENEFITS) ;  
QUIT;
```

The IN Condition (continued)



The resulting output:

ANN	BECKER	33281
CHRIS	DOBSON	33281
ALLEN	PARK	31373
BETTY	JOHNSON	31373

The NOT IN Condition



Who are the employees and which stores do they work for that have *not* had an insurance claim?

```
PROC SQL;  
  SELECT FNAME, LNAME, STORENO  
  FROM EMPLOYEE  
  WHERE COMPRESS(FNAME !! LNAME) NOT IN  
    (SELECT COMPRESS(FNAME !! LNAME)  
     FROM BENEFITS) ;  
QUIT;
```

The NOT IN Condition (continued)



The resulting output:

FNAME	LNAME	STORENO
-----	-----	-----
EARL	FISHER	33281
GARY	HOWE	33281
JACK	KELLER	33312
LARRY	MOORE	33312
NANCY	PAUL	33312
RICK	TENNY	33312
VIC	WATSON	31983
ARNIE	CARLSON	31983
DAVID	GELDER	31983
HARRY	JACKSON	31983
KELLY	LARSON	31983
MARY	NELSON	31381
PAULA	RILEY	31381

Checking Multiple Values in the Subquery



Select employees that had a claim over 1000.

```
PROC SQL;  
  SELECT *  
  FROM EMPLOYEE  
  WHERE COMPRESS(LNAME !! FNAME) IN  
        (SELECT COMPRESS(LNAME !! FNAME)  
         FROM BENEFITS  
         WHERE CLAIMS > 1000) ;  
QUIT;
```

The resulting output:

FNAME	LNAME	STORENO
ANN	BECKER	33281
ALLEN	PARK	31373
BETTY	JOHNSON	31373

Correlated Subqueries



- The previous subqueries have been simple subqueries that are self-contained and that execute independently of the outer query.
- A correlated subquery requires a value or values to be passed to it by the outer query.
- After the subquery runs, it passes the results back to the outer query.
- Correlated subqueries can return single or multiple values.

Source: PROC SQL documentation.

Correlated Subquery Example



```
proc sql;
  title 'Oil Reserves of Countries in Africa';
  select * from sql.oilrsrvs o
  where 'Africa' =
    (select Continent from sql.countries c where c.Name = o.Country);
Quit;
```

Processing:

- The outer query selects the first row from the OILRSRVS table and then passes the value of the Country column, Algeria , to the subquery.
- At this point, the subquery internally looks like this:

```
(select Continent from sql.countries c where c.Name = 'Algeria');
```

- The subquery selects that country from the COUNTRIES table.
- Subquery then passes the country's continent back to the WHERE clause in the outer query.
- If the continent is Africa, then the country is selected and displayed.
- The outer query then selects each subsequent row from the OILRSRVS table and passes the individual values of Country to the subquery.
- The subquery returns the appropriate values of Continent to the outer query for comparison in its WHERE clause.

Correlated Subquery Output



Oil Reserves of Countries in Africa

Country	Barrels
-----	-----
Algeria	9,200,000,000
Egypt	4,000,000,000
Gabon	1,000,000,000
Libya	30,000,000,000
Nigeria	16,000,000,000

Subqueries on Multiple Tables



Select customers with no purchase in last six months.

```
PROC SQL;  
  CREATE TABLE Nopurch_last180_w as  
  SELECT *  
  FROM Customers  
  WHERE Custid NOT IN  
    (SELECT Custid FROM Orders  
     WHERE Today() - Odate le 180 ) ;  
Quit;  
Proc Print Data=Nopurch_last180_w;  
Run;
```

Obs	State	Countrycode	Custid
1	WI	US	1236

Subqueries Against Different Data Sets



Select store names, state that had over \$20,000 sales in February?

```
PROC SQL;  
    SELECT STATE, STORENAM, STORENO  
    FROM USSALES  
    WHERE STORENO IN  
        (SELECT STORENO  
        FROM FEBSALES  
        WHERE SALES > 20000) ;  
QUIT;
```

STATE	STORENAM	STORENO
MI	WOODBIDGE GROCERS	33281
IL	OAKRIDGE GROCERY STORE	31983
IL	VICTOR'S FOOD CENTRE	31381
IL	SAVE U MONEY	31373

- A subquery can only contain one variable on the SELECT statement.

The PROC SQL PASS-Through Facility



FROM subqueries are “Passed Through” to the DBMS on the FROM expression. The results set is processed by SAS in the outer query.

Example: Go to Teradata and pull party_id and total paid by that party.

```
PROC SQL INOBS=100;
  CONNECT TO TERADATA(USER=userid PASSWORD=password
                     TDPID=DTDATA1A);
  SELECT PARTY_ID,
         TOTPAID
  FROM CONNECTION TO TERADATA
    (SELECT PARTY_ID,
           SUM(TOTALPAID_AMT) AS TOTPAID
    FROM CUSTOMER_SUMMARY
    GROUP BY PARTY_ID
    FROM CUSTOMER_SUMMARY);
QUIT;
```

The PROC SQL PASS-Through Facility



List PARTY_ID , TOT_PAID for customers with TOTALPAID_AMT less than average TOTALPAID_AMT of all customers.

```
PROC SQL INOBS=100;
  CONNECT TO TERADATA(USER=userid PASSWORD=password
    TDPID=DTDATA1A);
  SELECT PARTY_ID,
    TOTPAID
  FROM CONNECTION TO TERADATA
    (SELECT PARTY_ID, SUM(TOTALPAID_AMT) AS TOTPAID
      FROM CUSTOMER_SUMMARY
      GROUP BY PARTY_ID
      HAVING SUM(TOTALPAID_AMT) <
        (SELECT AVG(TOTALPAID_AMT)
          FROM CUSTOMER_SUMMARY));
QUIT;
```

Note: One subquery is nested in another.

The PROC SQL PASS-Through Facility



List FIRST_NAME, LAST_NAME from INDIVIDUAL table all customers with TOTALPAID_AMT > \$90 in CUSTOMER_SUMMARY table. PARTY_ID is the common key between these two tables.

```
PROC SQL INOBS=100;
  CONNECT TO TERADATA(USER=userid PASSWORD=password
    TDPID=DTDATA1A);
  SELECT FIRST_NAME, LAST_NAME
    FROM CONNECTION TO TERADATA
      (SELECT FIRST_NAME, LAST_NAME
        FROM INDIVIDUAL
        WHERE PARTY_ID IN
          (SELECT PARTY_ID,
            SUM(TOTALPAID_AMT)
            FROM CUSTOMER_SUMMARY
            HAVING SUM(TOTALPAID_AMT)>90));
QUIT;
```

The PROC SQL PASS-Through Facility



Repeat the previous query with a little different subquery.

```
PROC SQL INOBS=100;
  CONNECT TO TERADATA(USER=userid PASSWORD=password
    TDPID=DTDATA1A);
  SELECT FIRST_NAME, LAST_NAME
    FROM CONNECTION TO TERADATA
      (SELECT FIRST_NAME, LAST_NAME
        FROM INDIVIDUAL
        WHERE PARTY_ID IN
          (SELECT PARTY_ID
            FROM CUSTOMER_SUMMARY
            WHERE TOTALPAID_AMT >90));
QUIT;
```

Select Expression Subqueries



Create a 'Y','N' flag if customer had purchase in the last six months

```
PROC SQL;
  CREATE TABLE Flagnopurch_last180 as
  SELECT C.* ,
         Case
           When (Custid in
                 (SELECT Custid FROM Orders
                  WHERE Today() - Odate le 180)
                )
           Then 'y'
           Else 'n'
         End as Pflag
  FROM Customers C ;
Quit;

Proc Print Data=Flagnopurch_last180;
  Title 'Flagnopurchlast180';
Run;
```

Select Expression Subqueries (continued)



The resulting output:

Flagnopurch_last180

Obs	Ctate	Countrycode	Custid	Pflag
1	WI	US	1234	y
2	IL	US	1235	y
3	WI	US	1236	n
4	OR	US	1237	y
5	VI	VI	1238	y

More Select Subqueries



Summarize all order amounts in the last year.

```
PROC SQL;
  CREATE TABLE Summary12months as
  SELECT C.Custid      ,
         (SELECT Sum(O.OrderAmt)
          FROM Orders  O
          WHERE Today() - O.date lt 365
           and  C.custid = O.custid )
         as TotalOrderAmt
  FROM Customers C;
Quit;

Proc Print Data=Summary12months;
  Title 'Summary12months';
Run;
```

More Select Subqueries



The resulting output.

Summary12months		
Obs	Custid	Total OrderAmt
1	1234	200020
2	1235	110010
3	1236	.
4	1237	100005
5	1238	100005

A WHERE Subquery



An Inner join shows only customers with order amounts.

```
PROC SQL;
  CREATE TABLE Summary_12months as
  SELECT Custid,
         Sum(O.OrderAmt) as TotalOrderAmt
  FROM Orders O
       WHERE Custid in (SELECT C.Custid
                        FROM Customers C INNER JOIN
                        Orders O
                        on C.Custid = O.Custid )
         AND Today() - Odate lt 365
  GROUP BY custid;
Quit;
Proc Print Data=Summary_12months;
  Title 'Summary_12months';
Run;
```

A WHERE Subquery



The resulting output.

Summary_12months		
Obs	Custid	Total OrderAmt
1	1234	200020
2	1235	110010
3	1237	100005
4	1238	100005

Inline View



A subquery on the FROM expression is called an INLINE view.

```
PROC SQL;  
  CREATE TABLE Summary12months_i as  
  SELECT C.Custid,  
         O.TotalOrderAmt  
  FROM (select custid,  
           sum(orderamt) as TotalOrderAmt  
        from orders  
        WHERE Today() - Odate lt 365  
        group by custid  ) as o,  
        Customers as C  
  where C.custid = O.custid;  
Quit;  
  
Proc Print Data=Summary12months_i;  
  Title 'Summary12months_i';  
Run;
```

Inline View



The resulting output.

Summary12months_i		
Obs	Custid	Total OrderAmt
1	1234	200020
2	1235	110010
3	1237	100005
4	1238	100005

Standard SQL Joining



The previous results can be gotten with standard joining.

```
PROC SQL;  
CREATE TABLE Summary12months_j as  
SELECT C.Custid,  
       sum(o.orderamt) as TotalOrderAmt  
FROM orders o,  
       Customers as C  
where C.custid = O.custid  
       and Today() - Odate lt 365  
group by c. custid  
;  
Quit;
```

```
Proc Print Data=Summary12months_j;  
  Title 'Summary12months_j';  
Run;
```

Standard SQL Joining



The resulting output.

Summary12months_j		
Obs	Custid	Total OrderAmt
1	1234	200020
2	1235	110010
3	1237	100005
4	1238	100005

SAS Merging



PROC SORT and a DATA step can also produce the same results..

```
PROC proc sort data=orders;
  by custid; WHERE Today() - Odate lt 365;
run;
proc sort data=customers; by custid;
run;
data mergeds(keep=custid totalorderamt);
  merge orders(in=o)
        customers(in=c);
  by custid;
  if o and c;
  if first.custid then
    totalorderamt=0;
  totalorderamt+orderamt;
  put _all_;
  if last.custid;
run;
Proc Print Data=mergeds;
  Title 'Mergeds';
Run;
```



The resulting output.

Mergeds		
Obs	Custid	totalorderamt
1	1234	200020
2	1235	110010
3	1237	100005
4	1238	100005

Combining a Join with a Subquery



Example:

You want the city nearest to each city in the USCITYCOORDS table.

The query must:

- first select a city A
- compute the distance from city A to every other city
- finally select the city with the minimum distance from city A.
- This can be done by joining the USCITYCOORDS table to itself (self-join) and then determining the closest distance between cities by using another self-join in a subquery.

The following example is explained in detail in the PROC SQL documentation.

Combining a Join with a Subquery



The resulting output:

```
proc sql outobs=10;
  title 'Neighboring Cities';
  select a.City format=$10., a.State,
         a.Latitude 'Lat', a.Longitude 'Long',
         b.City format=$10., b.State,
         b.Latitude 'Lat', b.Longitude 'Long',
         sqrt(((b.latitude-a.latitude)**2) +
              ((b.longitude-a.longitude)**2)) as dist format=6.1
  from sql.uscitycoords a, sql.uscitycoords b
 where a.city ne b.city and
        calculated dist =
        (select min(sqrt(((d.latitude-
              c.latitude)**2) + ((d.longitude-c.longitude)**2)))
         from sql.uscitycoords c,
              sql.uscitycoords d
         where c.city = a.city and
              c.state = a.state and
              d.city ne c.city)
 order by a.city;
```

Combining a Join with a Subquery



The resulting output.

Neighboring Cities

City	State	Lat	Long	City	State	Lat	Long	dist
Albany	NY	43	-74	Hartford	CT	42	-73	1.4
Albuquerque	NM	36	-106	Santa Fe	NM	36	-106	0.0
Amarillo	TX	35	-102	Carlsbad	NM	32	-104	3.6
Anchorage	AK	61	-150	Nome	AK	64	-165	15.3
Annapolis	MD	39	-77	Washington	DC	39	-77	0.0
Atlanta	GA	34	-84	Knoxville	TN	36	-84	2.0
Augusta	ME	44	-70	Portland	ME	44	-70	0.0
Austin	TX	30	-98	San Antonio	TX	29	-98	1.0
Baker	OR	45	-118	Lewiston	ID	46	-117	1.4
Baltimore	MD	39	-76	Dover	DE	39	-76	0.0

Combining a Join with a Subquery



Process:

- Outer query joins the table to itself and finds distance between first city A1 in table A and city B2 (the first city not equal to city A1) in Table B.
- PROC SQL then runs the subquery.
- The subquery does another self-join and calculates min distance between city A1 and all other cities in the table other than city A1.
- The outer query tests to see whether the distance between cities A1 and B2 = minimum distance that was calculated by the subquery.
- If they are equal, then a row with cities A1 and B2, coordinates and distance is written.

When to Use Subqueries Versus Joins



What is the difference between the subqueries and joins?

- If you need data from more than one table, you must join them.
- If you need to combine different related rows in a single table, the table can be joined with itself.
- Use subqueries when the result you want requires more than one query and each subquery provides a subset of the table involved in the query.
- If a membership question is asked, then a subquery is usually used.
- EXISTS or NOT EXISTS operates only in a subquery.
- Some subqueries will be changed to a join by the SQL optimizer.
- Many queries can be written as either a subquery or a join.
- Generally, the join will be more efficient because a subquery is unable to directly apply the WHERE condition.

Additional Examples



Create a “Dashboard” table with multiple statistics on one row.

```
proc sql;
  create table dashboard as
  select distinct
    (select sum(amount)
     from sales
     where salesid='900009'
    ) as sum900009 format=comma8.2,
    (select avg(amount)
     from sales
     where salesid='900009'
    ) as avg900009 format=comma8.2,
    (select sum(amount)
     from sales
     where salesid='900386'
    ) as sum900386 format=comma8.2,
    (select avg(amount)
     from sales
     where salesid='900386'
    ) as avg900386 format=comma8.2
  from sales
  ;
quit;
```

Additional Examples



The “Dashboard” output.

dashboard				
Obs	sum900009	avg900009	sum900386	avg900386
1	52566.00	5,256.60	46630.00	5,181.11

Additional Examples



Use a SQL query to display all columns if CUSTID is duplicated.

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

```
337 PROC SQL;  
338   SELECT *  
339   FROM CONTACTS  
340   GROUP BY CUSTID  
341   HAVING COUNT(*) > 1  
342   order by custid, salesid, date;
```

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

```
343 QUIT;
```

Notes: The program worked correctly. Is note important?

Additional Examples



The resulting report.

Duplicated Custids by Query		
CUSTID	SALESID	DATE
10006	900222	01/05/99
10006	900222	02/04/99
10006	900489	05/04/99
10010	900009	04/02/99
10010	900009	04/05/99
10010	900222	03/02/99
10010	900222	04/04/99
10010	900386	02/04/99
10010	900489	05/03/99
10015	900009	01/04/99
10015	900222	01/03/99
10015	900222	01/04/99
10017	900009	01/04/99
10017	900045	02/05/99
10017	900489	01/04/99
10021	900009	01/02/99
10021	900201	04/02/99

Additional Examples



Coding as a subquery, eliminates the remerging message.

```
321 PROC SQL;
322     SELECT *
323     FROM CONTACTS
324     where custid in
325         (SELECT CUSTID
326          FROM CONTACTS
327          GROUP BY CUSTID
328          HAVING COUNT(*) > 1)
329     order by custid, salesid, date;
330     QUIT;
NOTE: PROCEDURE SQL used (Total process time):
```

Additional Examples



The resulting report is identical.

Duplicated Custids by Subquery		
CUSTID	SALESID	DATE
10006	900222	01/05/99
10006	900222	02/04/99
10006	900489	05/04/99
10010	900009	04/02/99
10010	900009	04/05/99
10010	900222	03/02/99
10010	900222	04/04/99
10010	900386	02/04/99
10010	900489	05/03/99
10015	900009	01/04/99
10015	900222	01/03/99
10015	900222	01/04/99
10017	900009	01/04/99
10017	900045	02/05/99
10017	900489	01/04/99
10021	900009	01/02/99
10021	900201	04/02/99

Conclusions



- PROC SQL subqueries provide more methods for joining, row selection, and much, much more.
- Like with any programming language, experience will show more and more value in subqueries.
- Remembering that subqueries usually return a single value makes understanding easier.
- Subqueries that return multiple values are compared to a single value with IN, ANY, ALL.
- Most subqueries are on WHERE, HAVING, but can appear other places.
- Performance needs to be benchmarked.
- Good commenting and documentation is crucial however code is written.



SYSTEMS SEMINAR CONSULTANTS, INC.

SAS® Training, Consulting, & Help Desk Services
2997 Yarmouth Greenway Drive • Madison, WI 53711
(608) 278-9964 • Fax (608) 278-0065
www.sys-seminar.com



Steve First

President

sfirst@sys-seminar.com

