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
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.

<table>
<thead>
<tr>
<th>SAS</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>table</td>
</tr>
<tr>
<td>variable</td>
<td>column</td>
</tr>
<tr>
<td>observation</td>
<td>row</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Division</th>
<th>Years</th>
<th>Sales</th>
<th>Expense</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHRIS</td>
<td>H</td>
<td>2</td>
<td>233.11</td>
<td>94.12</td>
<td>WI</td>
</tr>
<tr>
<td>MARK</td>
<td>H</td>
<td>5</td>
<td>298.12</td>
<td>52.65</td>
<td>WI</td>
</tr>
<tr>
<td>SARAH</td>
<td>S</td>
<td>6</td>
<td>301.21</td>
<td>65.17</td>
<td>MN</td>
</tr>
<tr>
<td>PAT</td>
<td>H</td>
<td>4</td>
<td>4009.21</td>
<td>322.12</td>
<td>IL</td>
</tr>
<tr>
<td>JOHN</td>
<td>H</td>
<td>7</td>
<td>678.43</td>
<td>150.11</td>
<td>WI</td>
</tr>
<tr>
<td>WILLIAM</td>
<td>H</td>
<td>11</td>
<td>3231.75</td>
<td>644.55</td>
<td>MN</td>
</tr>
<tr>
<td>ANDREW</td>
<td>S</td>
<td>24</td>
<td>1762.11</td>
<td>476.13</td>
<td>MN</td>
</tr>
<tr>
<td>BENJAMIN</td>
<td>S</td>
<td>3</td>
<td>201.11</td>
<td>25.21</td>
<td>IL</td>
</tr>
</tbody>
</table>
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


• 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
<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>10103.23</td>
<td>505.1615</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>455.1615</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
<td>751.6055</td>
</tr>
<tr>
<td>MI</td>
<td>33209.23</td>
<td>1660.462</td>
</tr>
<tr>
<td>MI</td>
<td>20132.43</td>
<td>1006.622</td>
</tr>
<tr>
<td>IL</td>
<td>20338.12</td>
<td>1016.906</td>
</tr>
<tr>
<td>IL</td>
<td>10332.11</td>
<td>516.6055</td>
</tr>
<tr>
<td>IL</td>
<td>32083.22</td>
<td>1604.161</td>
</tr>
<tr>
<td>IL</td>
<td>22223.12</td>
<td>1111.156</td>
</tr>
</tbody>
</table>
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;

WOW! Only ONE Semi-Colon!!

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

Group rows for summarization
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;
```

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

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

```sql
proc sql double;
    SELECT substr(storeno,1,2) as region
         label='Region of Store',
         sum(sales)
         format=dollar12.
    FROM ussales
    GROUP BY region;
quit;
```
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
Use the Macro Variable in a WHERE statement.

Example:

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

* COULD ALSO USE IN SQL QUERY;
PROC SQL;
SELECT *
  FROM NAMES
  WHERE NUMBER IN (&MINCODES);
QUIT;
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
## REPORT OF THE U.S. SALES

<table>
<thead>
<tr>
<th>STATE</th>
<th>AMOUNT OF SALES</th>
<th>5% TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>$10,103.23</td>
<td>$505.16</td>
</tr>
<tr>
<td>WI</td>
<td>$9,103.23</td>
<td>$455.16</td>
</tr>
<tr>
<td>WI</td>
<td>$15,032.11</td>
<td>$751.61</td>
</tr>
<tr>
<td>MI</td>
<td>$33,209.23</td>
<td>1660.46</td>
</tr>
</tbody>
</table>

PREPARED BY THE MARKETING DEPT.
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
<table>
<thead>
<tr>
<th>STATE</th>
<th>SALESCAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>AVG</td>
</tr>
<tr>
<td>WI</td>
<td>LOW</td>
</tr>
<tr>
<td>WI</td>
<td>HIGH</td>
</tr>
<tr>
<td>MI</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>MI</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>IL</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>IL</td>
<td>AVG</td>
</tr>
<tr>
<td>IL</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>IL</td>
<td>VERY HIGH</td>
</tr>
</tbody>
</table>
**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;

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

<table>
<thead>
<tr>
<th>STATE</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>84976.57</td>
</tr>
<tr>
<td>MI</td>
<td>53341.66</td>
</tr>
<tr>
<td>WI</td>
<td>34238.57</td>
</tr>
</tbody>
</table>
Subsetting Using the WHERE Clause

Select only specified rows for the output.

**Character**

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

**Compound**

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

**Numeric**

```
SELECT *
FROM USSALES
WHERE NSTATE IN (10, 20 ,30);
```
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)
PROC SQL;
SELECT STATE, STORENO, SUM(SALES) AS TOTSALES
FROM USSALES
GROUP BY STATE
WHERE TOTSALES > 500;
QUIT;

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.

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;

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENO</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>31212</td>
<td>10332.11</td>
</tr>
<tr>
<td>IL</td>
<td>31373</td>
<td>22223.12</td>
</tr>
<tr>
<td>IL</td>
<td>31381</td>
<td>32083.22</td>
</tr>
<tr>
<td>IL</td>
<td>31983</td>
<td>20338.12</td>
</tr>
<tr>
<td>MI</td>
<td>33281</td>
<td>33209.23</td>
</tr>
</tbody>
</table>

Notes:
• To subset data when grouping is in effect, HAVING must be used
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.
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;

<table>
<thead>
<tr>
<th>Obs</th>
<th>STATE</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IL</td>
<td>84976.57</td>
</tr>
<tr>
<td>2</td>
<td>MI</td>
<td>53341.66</td>
</tr>
<tr>
<td>3</td>
<td>WI</td>
<td>34238.57</td>
</tr>
</tbody>
</table>

Notes:

• When a CREATE statement is used in conjunction with a SELECT statement, a report will not be generated.
PROC SQL;
CREATE TABLE SUMSALE AS
    SELECT STATE,
           SUM(SALES) AS TOTSALES
    FROM USSALES
    GROUP BY STATE;

NOTE: Table WORK.SUMSALE created, with 3 rows and 2 columns.
QUIT;

PROC PRINT DATA=SUMSALE;
RUN;

NOTE: Procedure PRINT used:
real time 0.10 seconds
cpu time 0.00 seconds
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?

```sql
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:

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

<table>
<thead>
<tr>
<th>STATE</th>
<th>AVGSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>21244.14</td>
</tr>
<tr>
<td>MI</td>
<td>26670.83</td>
</tr>
</tbody>
</table>
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.

```sql
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:

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENO</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>32331</td>
<td>8103.23</td>
</tr>
<tr>
<td>IL</td>
<td>31212</td>
<td>8332.11</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENO</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>32331</td>
<td>8103.23</td>
</tr>
<tr>
<td>IL</td>
<td>31212</td>
<td>8332.11</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>9103.23</td>
</tr>
<tr>
<td>WI</td>
<td>8103.23</td>
</tr>
<tr>
<td>WI</td>
<td>10103.23</td>
</tr>
<tr>
<td>WI</td>
<td>13032.11</td>
</tr>
<tr>
<td>IL</td>
<td>25338.12</td>
</tr>
<tr>
<td>IL</td>
<td>8332.11</td>
</tr>
<tr>
<td>IL</td>
<td>30083.22</td>
</tr>
<tr>
<td>IL</td>
<td>26223.12</td>
</tr>
</tbody>
</table>
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.

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

The resulting output:

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>9103.23</td>
</tr>
<tr>
<td>WI</td>
<td>8103.23</td>
</tr>
<tr>
<td>WI</td>
<td>10103.23</td>
</tr>
<tr>
<td>WI</td>
<td>13032.11</td>
</tr>
<tr>
<td>IL</td>
<td>25338.12</td>
</tr>
<tr>
<td>IL</td>
<td>8332.11</td>
</tr>
<tr>
<td>IL</td>
<td>30083.22</td>
</tr>
<tr>
<td>IL</td>
<td>26223.12</td>
</tr>
</tbody>
</table>
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?

```sql
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Last Name</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>BECKER</td>
<td>33281</td>
</tr>
<tr>
<td>CHRIS</td>
<td>DOBSON</td>
<td>33281</td>
</tr>
<tr>
<td>ALLEN</td>
<td>PARK</td>
<td>31373</td>
</tr>
<tr>
<td>BETTY</td>
<td>JOHNSON</td>
<td>31373</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>STORENO</th>
</tr>
</thead>
<tbody>
<tr>
<td>EARL</td>
<td>FISHER</td>
<td>33281</td>
</tr>
<tr>
<td>GARY</td>
<td>HOWE</td>
<td>33281</td>
</tr>
<tr>
<td>JACK</td>
<td>KELLER</td>
<td>33312</td>
</tr>
<tr>
<td>LARRY</td>
<td>MOORE</td>
<td>33312</td>
</tr>
<tr>
<td>NANCY</td>
<td>PAUL</td>
<td>33312</td>
</tr>
<tr>
<td>RICK</td>
<td>TENNY</td>
<td>33312</td>
</tr>
<tr>
<td>VIC</td>
<td>WATSON</td>
<td>31983</td>
</tr>
<tr>
<td>ARNIE</td>
<td>CARLSON</td>
<td>31983</td>
</tr>
<tr>
<td>DAVID</td>
<td>GELDER</td>
<td>31983</td>
</tr>
<tr>
<td>HARRY</td>
<td>JACKSON</td>
<td>31983</td>
</tr>
<tr>
<td>KELLY</td>
<td>LARSON</td>
<td>31983</td>
</tr>
<tr>
<td>MARY</td>
<td>NELSON</td>
<td>31381</td>
</tr>
<tr>
<td>PAULA</td>
<td>RILEY</td>
<td>31381</td>
</tr>
</tbody>
</table>
Checking Multiple Values in the Subquery

Select employees that had a claim over 1000.

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

The resulting output:

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>STORENO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>BECKER</td>
<td>33281</td>
</tr>
<tr>
<td>ALLEN</td>
<td>PARK</td>
<td>31373</td>
</tr>
<tr>
<td>BETTY</td>
<td>JOHNSON</td>
<td>31373</td>
</tr>
</tbody>
</table>
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

```sql
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:

  ```sql
  (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

<table>
<thead>
<tr>
<th>Country</th>
<th>Barrels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algeria</td>
<td>9,200,000,000</td>
</tr>
<tr>
<td>Egypt</td>
<td>4,000,000,000</td>
</tr>
<tr>
<td>Gabon</td>
<td>1,000,000,000</td>
</tr>
<tr>
<td>Libya</td>
<td>30,000,000,000</td>
</tr>
<tr>
<td>Nigeria</td>
<td>16,000,000,000</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>Obs</th>
<th>State</th>
<th>Countrycode</th>
<th>Custid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WI</td>
<td>US</td>
<td>1236</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENAM</th>
<th>STORENO</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI</td>
<td>WOODBRIDGE GROCERS</td>
<td>33281</td>
</tr>
<tr>
<td>IL</td>
<td>OAKRIDGE GROCERY STORE</td>
<td>31983</td>
</tr>
<tr>
<td>IL</td>
<td>VICTOR'S FOOD CENTRE</td>
<td>31381</td>
</tr>
<tr>
<td>IL</td>
<td>SAVE U MONEY</td>
<td>31373</td>
</tr>
</tbody>
</table>

• A subquery can only contain one variable on the SELECT statement.
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;
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.
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
    TDPIID=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;
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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Cstate</th>
<th>Countrycode</th>
<th>Custid</th>
<th>Pflag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WI</td>
<td>US</td>
<td>1234</td>
<td>y</td>
</tr>
<tr>
<td>2</td>
<td>IL</td>
<td>US</td>
<td>1235</td>
<td>y</td>
</tr>
<tr>
<td>3</td>
<td>WI</td>
<td>US</td>
<td>1236</td>
<td>n</td>
</tr>
<tr>
<td>4</td>
<td>OR</td>
<td>US</td>
<td>1237</td>
<td>y</td>
</tr>
<tr>
<td>5</td>
<td>VI</td>
<td>VI</td>
<td>1238</td>
<td>y</td>
</tr>
</tbody>
</table>
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() - Odate 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.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Custid</th>
<th>Total OrderAmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1234</td>
<td>200020</td>
</tr>
<tr>
<td>2</td>
<td>1235</td>
<td>110010</td>
</tr>
<tr>
<td>3</td>
<td>1236</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1237</td>
<td>100005</td>
</tr>
<tr>
<td>5</td>
<td>1238</td>
<td>100005</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Custid</th>
<th>OrderAmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1234</td>
<td>200020</td>
</tr>
<tr>
<td>2</td>
<td>1235</td>
<td>110010</td>
</tr>
<tr>
<td>3</td>
<td>1237</td>
<td>100005</td>
</tr>
<tr>
<td>4</td>
<td>1238</td>
<td>100005</td>
</tr>
</tbody>
</table>
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;
The resulting output.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Custid</th>
<th>OrderAmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1234</td>
<td>200020</td>
</tr>
<tr>
<td>2</td>
<td>1235</td>
<td>110010</td>
</tr>
<tr>
<td>3</td>
<td>1237</td>
<td>100005</td>
</tr>
<tr>
<td>4</td>
<td>1238</td>
<td>100005</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Summary12months_j</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Obs</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
PROC SORT and a DATA step can also produce the same results.

```sas
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;
```
SAS Merging

The resulting output.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Custid</th>
<th>totalorderamt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1234</td>
<td>200020</td>
</tr>
<tr>
<td>2</td>
<td>1235</td>
<td>110010</td>
</tr>
<tr>
<td>3</td>
<td>1237</td>
<td>100005</td>
</tr>
<tr>
<td>4</td>
<td>1238</td>
<td>100005</td>
</tr>
</tbody>
</table>
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:

```sql
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.

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

<table>
<thead>
<tr>
<th>Obs</th>
<th>sum900009</th>
<th>avg900009</th>
<th>sum900386</th>
<th>avg900386</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>52566.00</td>
<td>5,256.60</td>
<td>46630.00</td>
<td>5,181.11</td>
</tr>
</tbody>
</table>
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?
The resulting report.

<table>
<thead>
<tr>
<th>CUSTID</th>
<th>SALESID</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10006</td>
<td>900222</td>
<td>01/05/99</td>
</tr>
<tr>
<td>10006</td>
<td>900222</td>
<td>02/04/99</td>
</tr>
<tr>
<td>10006</td>
<td>900489</td>
<td>05/04/99</td>
</tr>
<tr>
<td>10010</td>
<td>900009</td>
<td>04/02/99</td>
</tr>
<tr>
<td>10010</td>
<td>900009</td>
<td>04/05/99</td>
</tr>
<tr>
<td>10010</td>
<td>900222</td>
<td>03/02/99</td>
</tr>
<tr>
<td>10010</td>
<td>900222</td>
<td>04/04/99</td>
</tr>
<tr>
<td>10010</td>
<td>900386</td>
<td>02/04/99</td>
</tr>
<tr>
<td>10010</td>
<td>900489</td>
<td>05/03/99</td>
</tr>
<tr>
<td>10015</td>
<td>900009</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10015</td>
<td>900222</td>
<td>01/03/99</td>
</tr>
<tr>
<td>10015</td>
<td>900222</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10017</td>
<td>900009</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10017</td>
<td>900045</td>
<td>02/05/99</td>
</tr>
<tr>
<td>10017</td>
<td>900489</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10021</td>
<td>900009</td>
<td>01/02/99</td>
</tr>
<tr>
<td>10021</td>
<td>900201</td>
<td>04/02/99</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>CUSTID</th>
<th>SALESID</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10006</td>
<td>900222</td>
<td>01/05/99</td>
</tr>
<tr>
<td>10006</td>
<td>900222</td>
<td>02/04/99</td>
</tr>
<tr>
<td>10006</td>
<td>900489</td>
<td>05/04/99</td>
</tr>
<tr>
<td>10010</td>
<td>900009</td>
<td>04/02/99</td>
</tr>
<tr>
<td>10010</td>
<td>900009</td>
<td>04/05/99</td>
</tr>
<tr>
<td>10010</td>
<td>900222</td>
<td>03/02/99</td>
</tr>
<tr>
<td>10010</td>
<td>900222</td>
<td>04/04/99</td>
</tr>
<tr>
<td>10010</td>
<td>900386</td>
<td>02/04/99</td>
</tr>
<tr>
<td>10010</td>
<td>900489</td>
<td>05/03/99</td>
</tr>
<tr>
<td>10015</td>
<td>900009</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10015</td>
<td>900222</td>
<td>01/03/99</td>
</tr>
<tr>
<td>10015</td>
<td>900222</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10017</td>
<td>900009</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10017</td>
<td>900045</td>
<td>02/05/99</td>
</tr>
<tr>
<td>10017</td>
<td>900489</td>
<td>01/04/99</td>
</tr>
<tr>
<td>10021</td>
<td>900009</td>
<td>01/02/99</td>
</tr>
<tr>
<td>10021</td>
<td>900201</td>
<td>04/02/99</td>
</tr>
</tbody>
</table>
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.