AN INTRODUCTION TO THE SQL PROCEDURE
Chris Yindra, C. Y. Associates

Abstract
This tutorial will introduce the SQL (Structured Query Language) procedure through a series of simple examples. We will initially discuss choosing variables (SELECT) from SAS® data sets (FROM) where a specific criteria is met (WHERE). We will then discuss calculating and formatting values. Once the basic SQL syntax has been covered we will discuss more advanced features of SQL such as grouping and ordering data, selecting based on summary values, applying CASE logic and simple joins. Finally we will make a comparison of simple SQL queries to base SAS. This tutorial will provide attendees with all the tools necessary to write simple SQL queries. It is intended for SAS programmers who have no prior exposure to the SQL procedure as well as those new to SAS.

Introduction
The Structured Query Language (SQL) is a standardized language used to retrieve and update data stored in relational tables (or databases). When coding in SQL, the user is not required to know the physical attributes of the table such as data location and type. SQL is non-procedural. The purpose is to allow the programmer to focus on what data should be selected and not how to select the data. The method of retrieval is determined by the SQL optimizer, not by the user.

What is a table?
A table is a two dimensional representation of data consisting of columns and rows. In the SQL procedure a table can be a SAS data set, SAS data view, or table from a RDBMS. Tables are logically related by values such as a key column.

There are several implementations (versions) of SQL depending on the RDBMS being used. The SQL procedure supports most of the standard SQL. It also has many features that go beyond the standard SQL.

Terminology
The terminology used in SQL can be related to standard data processing terminology. A table in SQL is simply another term for a SAS data set or data view.

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>SAS</th>
<th>SQL equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>SAS dataset</td>
<td>Table</td>
</tr>
<tr>
<td>Record</td>
<td>Observation</td>
<td>Row</td>
</tr>
<tr>
<td>Field</td>
<td>Variable</td>
<td>Column</td>
</tr>
</tbody>
</table>

The table is where the data is stored. A row represents a particular entry. An employee may be represented as a row in a table. A column represents the particular values for all rows. Salary may be a column on a table. All employee's will have a value for salary.

Relational Tables
In SQL tables are logically related by a key column or columns.

<table>
<thead>
<tr>
<th>Dept</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCE</td>
<td>SMITH</td>
<td>23000</td>
</tr>
<tr>
<td>MCE</td>
<td>JONES</td>
<td>34000</td>
</tr>
<tr>
<td>INA</td>
<td>LEE</td>
<td>28000</td>
</tr>
<tr>
<td>INA</td>
<td>RAY</td>
<td>21000</td>
</tr>
</tbody>
</table>

Dept is the key column in Table 1 to join to the Dptcode key column in Table 2. Either Location or Manager in Table 2 could also act as a key into a third table.

Simple Queries
A query is merely a request for information from a table or tables. The query result is typically a report but can also be another table. For instance:

I would like to select last name, department, and salary from the employee table where the employee's salary is greater than 35,000.

How would this query (request) look in SQL?

```
SELECT LASTNAME, DEPARTMENT, SALARY
FROM CLASS.EMPLOY
WHERE SALARY GT 35000
```

Herein lies the simplicity of SQL. The programmer can focus on what they want and SQL will determine how to get it. The fundamental approach is

```
SELECT ...
FROM...
WHERE...
```
In SAS, queries are submitted with PROC SQL.

**Basic Syntax**

```sql
PROC SQL;
  SELECT column, column . . .
  FROM tablename|viewname . . .
QUIT;
```

- Statements (clauses) in the SQL procedure are not separated by semicolons, the entire query is terminated with a semicolon.
- Items in an SQL statement are separated by a comma.
- There is a required order of statements in a query.
- One SQL procedure can contain many queries and a query can reference the results from previous queries.
- The SQL procedure can be terminated with a QUIT statement, RUN statements have no effect.

**SELECT**

- To retrieve and display data a SELECT statement is used.
- The data will be displayed in the order you list the columns in the SELECT statement.
- A column can be a variable, calculated value or formatted value.
- An asterisk (*) can be used to select all columns.

**FROM**

- The FROM statement specifies the input table or tables.

Examples:

I would like to select the social security number, salary and bonus (columns) for all employee's (rows) from the employee table.

```sql
LIBNAME CLASS 'C:\CYDATA';
PROC SQL;
  SELECT SSN, SALARY, BONUS
  FROM CLASS.EMPLOYEE;
QUIT;
```

I would like to select all the information (columns) for all employee's (rows) on the employee table.

```sql
PROC SQL;
  SELECT *
  FROM CLASS.EMPLOYEE;
QUIT;
```

**Selecting Rows with a WHERE Clause**

The WHERE clause specifies rows to be selected for the query.

**WHERE expression1 [AND/OR] expression2**

Example:

I would like to select the social security number, salary and bonus (columns) from the employee table where the employee is in department GIO and has a salary less that 35,000.

```sql
PROC SQL;
  SELECT SSN, SALARY, BONUS
  FROM CLASS.EMPLOYEE
  WHERE DEPT = 'GIO' AND SALARY > 35000;
QUIT;
```

**WHERE Clause Operators**

The WHERE clause supports many comparison operators. It also supports logical NOTs and AND/OR to create compound expressions.

**Standard comparison operators**

- **EQ or =** Equal to
- **NE ^=** Not Equal To
- **GT >** Greater Than
- **GE >=** Greater Than or Equal To
- **LT <** Less Than
- **LE <=** Less Than or Equal To

**Special operators**

- **BETWEEN .. AND** Compare to a range
- **IN** Compare to a series of values
- **IS MISSING ( NULL)** Value is missing
- **LIKE** Compare to wildcards (% or _)
- **CONTAINS** Compare to a substring
- **=*** Sounds like
- **EXISTS** Compare to a subquery

**Logical operators**

- **AND** Compound WHERE clause
- **OR** Compound WHERE clause
- **NOT** Logical NOT
AND is resolved before OR. Use parenthesis to override this.

Example:

I would like to select the last name and social security number from the employee table where the employee is earning between 20,000 and 30,000 dollars in salary. Include a title stating that these employees are in a middle salary range.

TITLE 'EMPLOYEE'S IN THE MIDDLE SALARY RANGE';
PROC SQL;
  SELECT LASTNAME, DEPT, SALARY, BONUS
  FROM CLASS.EMPLOYEE
  WHERE SALARY BETWEEN 20000 AND 30000;
QUIT;

Some additional examples of WHERE clauses:

Show the people in job grades 10, 11 and 12
WHERE GRADE IN ('10','11','12')

Show the people in all departments beginning with G
WHERE DEPT LIKE 'G%'

Selects GIO, GPO, GPD etc.
Show the people with a missing hire date
WHERE HIREDT IS MISSING

ANDs and Ors can be used to create compound WHERE clauses.

Who is selected by the following query?

PROC SQL:
  SELECT LASTNAME, DEPT, SALARY
  FROM CLASS.EMPLOY
  WHERE DEPT EQ 'GPO' OR DEPT EQ 'GIO'
  AND SALARY GT 30000;
QUIT;

Output:

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>DEPT</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLYNN</td>
<td>GPO</td>
<td>32500</td>
</tr>
<tr>
<td>SILVER</td>
<td>GIO</td>
<td>31500</td>
</tr>
<tr>
<td>SILVAY</td>
<td>GPO</td>
<td>18600</td>
</tr>
<tr>
<td>BARBER</td>
<td>GPO</td>
<td>20800</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>GPO</td>
<td>24500</td>
</tr>
<tr>
<td>RYAN</td>
<td>GPO</td>
<td>23500</td>
</tr>
<tr>
<td>Brown</td>
<td>GIO</td>
<td>30500</td>
</tr>
<tr>
<td>GLYNN</td>
<td>GIO</td>
<td>39500</td>
</tr>
</tbody>
</table>

Why are there people with salaries less than 30,000 in this output?

Remember ANDs are resolved before Ors so we evaluate the AND expression first.

Is the person in department GIO AND have a salary greater than 30,000
We select all of those people and then evaluate the OR expression.

OR is this person in department GPO.
So the previous query selects everyone in GPO and only those people in GIO with salaries greater than 30,000.
We could use parenthesis to override.

WHERE (DEPT EQ 'GIO' OR DEPT EQ 'GPO') AND SALARY GT 30000

Because whatever is in parenthesis is evaluated first, this WHERE clause would select people from either department, however everyone selected will have a salary greater than 30,000.

Calculating and Formatting Values

New values can be calculated in the SELECT clause.

Example:

I would like to select an employee's last name, salary, bonus and also show the percent of the employee's bonus to their salary from the employee table where the employee is in department GPO.

PROC SQL:
  SELECT LASTNAME, SALARY, BONUS, BONUS / SALARY
  FROM CLASS.EMPLOY
  WHERE DEPT EQ 'GPO';
QUIT;

Output:

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>SALARY</th>
<th>BONUS</th>
<th>BONUS / SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLYNN</td>
<td>32500</td>
<td>1512</td>
<td>0.046532</td>
</tr>
<tr>
<td>SILVER</td>
<td>31500</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SILVAY</td>
<td>18600</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BARBER</td>
<td>20800</td>
<td>1000</td>
<td>0.048077</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>24500</td>
<td>1100</td>
<td>0.044898</td>
</tr>
<tr>
<td>RYAN</td>
<td>23500</td>
<td>2300</td>
<td>0.097872</td>
</tr>
</tbody>
</table>
The new calculated column can be given a name with AS.

```sql
PROC SQL;
  SELECT LASTNAME, SALARY, BONUS,
       BONUS / SALARY AS BONUSPR
  FROM CLASS.EMPLOY
  WHERE DEPT EQ 'GPO';
QUIT;
```

Output:

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>SALARY</th>
<th>BONUS</th>
<th>BONUSPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLYNN</td>
<td>32500</td>
<td>1512</td>
<td>0.046532</td>
</tr>
<tr>
<td>SILVAY</td>
<td>18600</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BARBER</td>
<td>20800</td>
<td>1000</td>
<td>0.048077</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>24500</td>
<td>1100</td>
<td>0.044898</td>
</tr>
<tr>
<td>RYAN</td>
<td>23500</td>
<td>2300</td>
<td>0.097872</td>
</tr>
</tbody>
</table>

Columns may also be calculated with SAS functions. SAS SQL supports the use of most data step functions in the select statement (LAG, DIF, SOUND are not supported). Summary functions using more than one variable operate on each row.

Example:

I would like to select an employee's last name, social security number and also show the employee's total compensation which is the sum of their salary and bonus from the employee table where the employee is in department GPO.

```sql
PROC SQL;
  SELECT LASTNAME, SSN,
       SUM(SALARY,BONUS) AS TOTCOMP
  FROM CLASS.EMPLOY
  WHERE DEPT EQ 'GPO';
QUIT;
```

Output:

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>SSN</th>
<th>TOTCOMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLYNN</td>
<td>010101010</td>
<td>34012</td>
</tr>
<tr>
<td>SILVAY</td>
<td>111111117</td>
<td>18600</td>
</tr>
<tr>
<td>BARBER</td>
<td>111111120</td>
<td>21800</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>222222226</td>
<td>25600</td>
</tr>
<tr>
<td>RYAN</td>
<td>222222227</td>
<td>25800</td>
</tr>
</tbody>
</table>

Example:

I would like to select an employee's last name and determine the whole number of years that they have been employed from the employee table where the employee is in department GPO. The years employed can be calculated as the current date minus the employee's hire date (days the employee has been employed) divided by 365.

```sql
PROC SQL;
  SELECT LASTNAME,
       INT((TODAY() - HIREDT) / 365) AS YRSEMPL
  FROM CLASS.EMPLOY
  WHERE DEPT EQ 'GPO';
QUIT;
```

Output:

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>YRSEMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLYNN</td>
<td>19</td>
</tr>
<tr>
<td>SILVAY</td>
<td>5</td>
</tr>
<tr>
<td>BARBER</td>
<td>3</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>12</td>
</tr>
<tr>
<td>RYAN</td>
<td>7</td>
</tr>
</tbody>
</table>

Selecting on a Calculated Column

When selecting on a calculated column, the CALCULATED keyword must be used in the WHERE clause.

Example:

Modify the previous query to select only those employees with over ten years of service.

```sql
PROC SQL;
  SELECT LASTNAME,
       INT((TODAY() - HIREDT) / 365) AS YRSEMPL
  FROM CLASS.EMPLOY
  WHERE DEPT EQ 'GPO' AND YRSEMPL GT 10;
QUIT;
```

This query results in the following error message:

```
ERROR: The following columns were not found in the contributing tables: YRSEMPL.
```
Instead, use the CALCULATED keyword:

```sql
PROC SQL;
    SELECT LASTNAME,
          INT((TODAY() - HIREDT) / 365) AS YRSEMPL
    FROM CLASS.EMPLOY
    WHERE DEPT EQ 'GPO' AND
          CALCULATED YRSEMPL GT 10
    ;
QUIT;
```

Output:

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>YRSEMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLYNN</td>
<td>18</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>11</td>
</tr>
</tbody>
</table>

Formatting Values

Formats can be used for any variable or calculated column in the select statement to format values on the output. Formats allow the programmer the ability to define the number of decimals to display, insert dollar signs and commas, decode values, etc. Any valid SAS or user-defined format can be used.

Example:

Display an employee's total compensation in DOLLAR format.

```sql
PROC SQL;
    SELECT LASTNAME, SSN,
          SUM(SALARY,BONUS) AS TOTCOMP
    FORMAT = DOLLAR8.
    FROM CLASS.EMPLOY
    WHERE DEPT EQ 'GPO'
    ;
QUIT;
```

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>SSN</th>
<th>TOTCOMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLYNN</td>
<td>010101010</td>
<td>$34,012</td>
</tr>
<tr>
<td>SILVAY</td>
<td>111111117</td>
<td>$18,600</td>
</tr>
<tr>
<td>BARBER</td>
<td>111111120</td>
<td>$21,800</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>222222226</td>
<td>$25,600</td>
</tr>
<tr>
<td>RYAN</td>
<td>222222227</td>
<td>$25,800</td>
</tr>
</tbody>
</table>

User written formats can also be used in the select statement.

Example:

I would like to select an employee's last name, job grade, and salary from the employee table where the employee is in job grades 10 or 12 and I would like to order the employee's on the report by descending salary.

```sql
PROC SQL;
    SELECT LASTNAME, GRADE, SALARY
    FROM CLASS.EMPLOY
    WHERE GRADE IN('10','12')
    ORDER BY SALARY DESC;
QUIT;
```
Case Logic

Case logic is used when you would like to assign specific values based on some criteria. It is used in place of conditional IF statements in SQL. The case statement can compare to a single value or an expression and returns a single value for each condition met.

Case logic is used when you would like to assign specific values based on some criteria. It is used in place of conditional IF statements in SQL. The case statement can compare to a single value or an expression and returns a single value for each condition met.

```
CASE [operand]
  WHEN [condition] THEN [result]
  [ WHEN condition THEN result] …
  [ELSE result]
END [AS column]
```

If a CASE operand is specified, the WHEN condition must be an equality. If a CASE operand is not specified, the WHEN condition must be a valid Boolean expression. The result can be a single value or another CASE statement (nested CASE).

If we wanted to assign a value based upon the fact that an employee was in department GIO we would use:

```
CASE DEPT
  WHEN GIO THEN assigned value
  END AS new column
```

Instead of comparing to a single value, we could also compare to an expression. For instance suppose we wanted to assign a value based upon the fact that that an employee was in department GIO and in job grade 10 we would use:

```
CASE
  WHEN DEPT = GIO AND GRADE = '10'
  THEN assigned value
  END AS new column
```

Example:

I would like to select an employee's last name and department. I would also like to assign a holiday bonus based upon an employee's department. I would also like to order the output by last name.

```
PROC SQL;
  SELECT LASTNAME, DEPT, GRADE,
    CASE DEPT
      WHEN 'GIO' THEN 100
      WHEN 'GPO' THEN 200
      WHEN 'IIO' THEN 300
      ELSE 0
    END AS HOLBON
  FROM CLASS.EMPLOY
  ORDER BY LASTNAME;
QUIT;
```

Output (partial):

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>DEPT</th>
<th>HOLBON</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARBER</td>
<td>GPO</td>
<td>200</td>
</tr>
<tr>
<td>BARNHART</td>
<td>GIO</td>
<td>100</td>
</tr>
<tr>
<td>BROWN</td>
<td>GIO</td>
<td>100</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>GPO</td>
<td>200</td>
</tr>
<tr>
<td>CROWLEY</td>
<td>GIO</td>
<td>100</td>
</tr>
<tr>
<td>CROWLEY</td>
<td>GIO</td>
<td>100</td>
</tr>
<tr>
<td>DAVIDSON</td>
<td>IIO</td>
<td>300</td>
</tr>
<tr>
<td>FERRIO</td>
<td>GIO</td>
<td>100</td>
</tr>
</tbody>
</table>

Example:

I would like to select an employee's last name, department and job grade. I would also like to assign a Holiday bonus. The Holiday bonus is based upon an employee's department and job grade I would like to include employee's where their department is equal to GIO or GPO.

```
PROC SQL;
  SELECT LASTNAME, DEPT, GRADE,
    CASE DEPT
      WHEN 'GIO' THEN
        CASE
          WHEN GRADE IN('10','11','12') THEN 100
          ELSE 300
        END
      WHEN 'GPO' THEN
        CASE GRADE
          WHEN '10' THEN 400
        END
    END AS HOLBON
  FROM CLASS.EMPLOY;
QUIT;
```
WHEN '11' THEN 500
ELSE 600
END
END AS HOLBON
FROM CLASS.EMPLOY
WHERE DEPT IN ('GIO','GPO');
QUIT;

Output (partial):

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>DEPT</th>
<th>GRADE</th>
<th>HOLBON</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLADSTONE</td>
<td>GIO</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>GLYNN</td>
<td>GPO</td>
<td>10</td>
<td>400</td>
</tr>
<tr>
<td>SILVER</td>
<td>GIO</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>BARNHART</td>
<td>GIO</td>
<td>12</td>
<td>100</td>
</tr>
<tr>
<td>SILVAY</td>
<td>GPO</td>
<td>12</td>
<td>600</td>
</tr>
<tr>
<td>BARBER</td>
<td>GPO</td>
<td>12</td>
<td>600</td>
</tr>
<tr>
<td>FERRIO</td>
<td>GIO</td>
<td>13</td>
<td>300</td>
</tr>
<tr>
<td>LOUDEN</td>
<td>GIO</td>
<td>13</td>
<td>300</td>
</tr>
<tr>
<td>SMITH</td>
<td>GIO</td>
<td>13</td>
<td>300</td>
</tr>
<tr>
<td>VERNLUND</td>
<td>GIO</td>
<td>13</td>
<td>300</td>
</tr>
<tr>
<td>CARPENTER</td>
<td>GPO</td>
<td>14</td>
<td>600</td>
</tr>
</tbody>
</table>

Summary Functions

Summary functions summarize column values for all rows in a table producing an aggregate value. Rows will be summarized to the lowest logical summary. If all columns on the SELECT statement are summary functions then the summary will be based upon all rows in the table.

Some common statistics that the SQL procedure supports are:

- **AVG, MEAN**: mean of values
- **COUNT, FREQ, N**: number of nonmissing values
- **NMISS**: number of missing values
- **MAX**: maximum value
- **MIN**: minimum value
- **RANGE**: range from MIN to MAX
- **SUM**: sum of values

Example:

I would like to show the average salary, the total salary, the minimum salary and maximum salary for the entire employee table.

PROC SQL;

```
SELECT AVG(SALARY) AS AVSAL,
       SUM(SALARY) AS SUMSAL,
       MIN(SALARY) AS MINSAL,
       MAX(SALARY) AS MAXSAL
FROM CLASS.EMPLOY;
QUIT;
```

Output:

<table>
<thead>
<tr>
<th>AVSAL</th>
<th>SUMSAL</th>
<th>MINSAL</th>
<th>MAXSAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>23273.91</td>
<td>535300</td>
<td>9500</td>
<td>41600</td>
</tr>
</tbody>
</table>

What is produced by the following query?

PROC SQL;

```
SELECT DEPT, SUM(SALARY), SUM(BONUS)
FROM CLASS.EMPLOY
WHERE DEPT = 'GIO';
```

Output: (partial):

<table>
<thead>
<tr>
<th>DEPT</th>
<th>SUM(SALARY)</th>
<th>SUM(BONUS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIO</td>
<td>315600</td>
<td>23854</td>
</tr>
<tr>
<td>GIO</td>
<td>315600</td>
<td>23854</td>
</tr>
<tr>
<td>GIO</td>
<td>315600</td>
<td>23854</td>
</tr>
</tbody>
</table>

When summary functions and detail values are used in the SELECT statement without a GROUP BY, the summary will be for the entire table although each detail row will be displayed. In this case we are seeing a row for each employee in department GIO with the total salary and total bonus for the entire department displayed.

Grouping Data

To summarize and display data in groups, use a GROUP BY clause with column functions.

```
GROUP BY column1 [,column2, ...]
```

To arrange results in a particular order (ascending or descending) use an ORDER BY statement.

Example:

I would like to select the department and summarize the average salary, total salary, minimum salary and maximum salary from the employee table. I would like to group the summary values by department and order the results by descending average salary.

PROC SQL;

```
SELECT DEPT, AVG(SALARY) AS AVSAL,
       SUM(SALARY) AS SUMSAL,
       MIN(SALARY) AS MINSAL,
       MAX(SALARY) AS MAXSAL
FROM CLASS.EMPLOY
WHERE DEPT = 'GIO';
QUIT;
```
FROM CLASS.EMPLOY
GROUP BY DEPT
ORDER BY AVSAL DESC;
QUIT;

Output:

<table>
<thead>
<tr>
<th>DEPT</th>
<th>AVSAL</th>
<th>SUMSAL</th>
<th>MINSAL</th>
<th>MAXSAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIO</td>
<td>24276</td>
<td>315600</td>
<td>9500</td>
<td>41600</td>
</tr>
<tr>
<td>GPO</td>
<td>23980</td>
<td>119900</td>
<td>18600</td>
<td>32500</td>
</tr>
<tr>
<td>RIO</td>
<td>21050</td>
<td>42100</td>
<td>18600</td>
<td>23500</td>
</tr>
<tr>
<td>FIN</td>
<td>20900</td>
<td>20900</td>
<td>20900</td>
<td>20900</td>
</tr>
<tr>
<td>IIO</td>
<td>18400</td>
<td>36800</td>
<td>18300</td>
<td>18500</td>
</tr>
</tbody>
</table>

With SQL we can also easily calculate values using subtotals. This requires that the summary values are remerged back into the detail table. Fortunately SQL handles all of this on its own.

Example:

I would like to select an employee's department and last name and calculate the employee's percent of salary against the department subtotal from the employee table. I would like to order the output by descending salary percent.

PROC SQL;
SELECT DEPT, LASTNAME,
SALARY / SUM(SALARY) AS PERCSAL
FORMAT = PERCENT6.
FROM CLASS.EMPLOY
GROUP BY DEPT
ORDER BY PERCSAL DESC;
QUIT;

Output (partial):

<table>
<thead>
<tr>
<th>DEPT</th>
<th>LASTNAME</th>
<th>PERCSAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIN</td>
<td>JACKSON</td>
<td>100%</td>
</tr>
<tr>
<td>RIO</td>
<td>RYAN</td>
<td>56%</td>
</tr>
<tr>
<td>IIO</td>
<td>JONES</td>
<td>50%</td>
</tr>
<tr>
<td>IIO</td>
<td>DAVIDSON</td>
<td>50%</td>
</tr>
<tr>
<td>RIO</td>
<td>WOOD</td>
<td>44%</td>
</tr>
<tr>
<td>GPO</td>
<td>GLYNN</td>
<td>27%</td>
</tr>
<tr>
<td>GPO</td>
<td>CARPENTER</td>
<td>20%</td>
</tr>
<tr>
<td>GPO</td>
<td>RYAN</td>
<td>20%</td>
</tr>
</tbody>
</table>

We can also choose to display rows by selecting on summary values. This requires the use of the HAVING expression. The HAVING expression follows the GROUP BY clause. The HAVING expression selects summary rows based on summary functions.

HAVING sql-expression

SQL-expression is any valid SQL expression. It is evaluated once for each group in the query. The HAVING expression can compare to detail rows, which will remerge detail rows with summary rows. If there is no GROUP BY clause, the comparison is based on a table summary.

Example:

I would like to select department and subtotal salary and bonus from the employee table. I would like to include only those departments with total salaries greater than $100,000. I would also like to order the output by descending total salary.

PROC SQL;
SELECT DEPT, SUM(SALARY) AS TOTSAL
FORMAT=DOLLAR11.2,
SUM(BONUS) AS TOTBON
FORMAT=DOLLAR11.2
FROM CLASS.EMPLOY
GROUP BY DEPT
HAVING TOTSAL > 100000
ORDER BY TOTSAL DESC;
QUIT;

Output:

<table>
<thead>
<tr>
<th>DEPT</th>
<th>TOTSAL</th>
<th>TOTBON</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIO</td>
<td>$315,600.00</td>
<td>$23,854.30</td>
</tr>
<tr>
<td>GPO</td>
<td>$119,900.00</td>
<td>$5,912.30</td>
</tr>
</tbody>
</table>

Joining Tables

Joining tables is a way of bringing rows from different tables together. Rows are usually joined on a key column or columns. If the value of the key(s) in both tables is equal, the rows are joined.

Types of Joins

A conceptual view of a join involves combining all rows from the contributing tables and then eliminate those that do not meet the WHERE criteria (equality on the keys).

The actual methodology used by the SQL procedure is not determined by the user but by the SQL Optimizer. This allows the user to focus on what they want logically and not on the internals of how to extract it.
There are two primary types of joins in SQL, an equi-join and a Cartesian join.

Cartesian joins- Do not use equality in the WHERE clause or do not use a WHERE clause. This forces each row of the first table to be combined with all rows from the second table if they meet the WHERE criteria.

Equi-joins- Use equality in the WHERE clause. Rows with matching key values are joined. The keys must be of the same length and data type.

Both types of joins allow columns from multiple tables to be included in one where clause. All contributing tables are listed in the FROM clause. Columns with the same names on one or more tables must be qualified by preceding the column name with the name of the contributing table (separated by a period).

Example:

I would like to select an employee's last name, location and manager from the employee table and the department file where the department value from the employee table is equal to the department value on the department table. Last name comes from the employee table. Location and manager come from the department file.

PROC SQL:
SELECT LASTNAME, LOCATION, MANAGER
FROM CYLIB.EMPLOY, CYLIB.DEPTFILE
WHERE EMPLOY.DEPT = DEPTFILE.DEPT
QUIT;

Output (partial):

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>LOCATION</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLADSTONE</td>
<td>WINDSOR</td>
<td>GARCIA</td>
</tr>
<tr>
<td>GLYNN</td>
<td>BOSTON</td>
<td>WIER</td>
</tr>
<tr>
<td>SILVER</td>
<td>WINDSOR</td>
<td>GARCIA</td>
</tr>
<tr>
<td>BARNHART</td>
<td>WINDSOR</td>
<td>GARCIA</td>
</tr>
<tr>
<td>DAVIDSON</td>
<td>NEW YORK</td>
<td>LESH</td>
</tr>
</tbody>
</table>

Joining more than two tables:

More than two tables can be joined in a single query. A different key can be used to join different tables.

Example:

We have three files that we would like to join. The PURCHASE file contains all invoice numbers and the department that made the purchase. The APAY (accounts payable file) contains a vendor code and invoice amount for each invoice. The VENDFILE contains a vendor code and relevant vendor information (name, phone).

<table>
<thead>
<tr>
<th>PURCHASE</th>
<th>APAY</th>
<th>VENDFILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPT</td>
<td>INVNO</td>
<td>VENDNUM</td>
</tr>
<tr>
<td>INVNOBR</td>
<td>INVAMT</td>
<td>VENDNAME</td>
</tr>
<tr>
<td>VENDOR</td>
<td></td>
<td>VENDOR</td>
</tr>
</tbody>
</table>

We would like to total the invoice amounts that each department owes each vendor. To do this we have to join the PURCHASE table to the APAY table on the invoice number and the APAY table to the VENDFILE on vendor number.

PROC SQL:
SELECT DEPT, VENDNAME, SUM(INVAMT) AS TOTAL
FROM CLASS.PURCHASE AS A,
     CLASS.APAY AS B,
     CLASS.VENDFILE AS C
WHERE A.INVNUMBR = B.INVNO
    AND B.VENDOR = C.VENDNUM
GROUP BY DEPT, VENDNAME;

Output (partial):

<table>
<thead>
<tr>
<th>DEPT</th>
<th>VENDNAME</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIO</td>
<td>CANTON COMPUTER CENTER</td>
<td>825</td>
</tr>
<tr>
<td>GIO</td>
<td>COUNTRY OFFICE SUPPLIES</td>
<td>475</td>
</tr>
<tr>
<td>GIO</td>
<td>OTTO PRINT SHOP</td>
<td>500</td>
</tr>
<tr>
<td>GIO</td>
<td>SAS INSTITUTE</td>
<td>3906</td>
</tr>
<tr>
<td>GPO</td>
<td>COUNTRY OFFICE SUPPLIES</td>
<td>900</td>
</tr>
</tbody>
</table>

Creating Tables

In all of the previous examples, the result of the query has been an output report. We could instead create a new table (SAS data set). The CREATE statement is used to create SQL tables as a permanent or temporary SAS data sets with the SELECT clause providing the variable list.

CREATE TABLE tablename AS
[SELECT list]

Example:

I would like to create a temporary table SENIORS by selecting an employee's last name, age and gender from the employee table where the employee's age is greater than 50. I would like to order the table by descending age.
PROC SQL;
CREATE TABLE WORK.SENIORS AS
SELECT LASTNAME, AGE, GENDER
FROM CLASS.EMPLOY
WHERE AGE GT 50
ORDER BY AGE DESC;

Comparing SQL With Base SAS

Many common programming tasks can be accomplished with either SQL or base SAS.

Example;

PROC SQL;
SELECT LASTNAME, SALARY, BONUS,
BONUS / SALARY AS BONUSPR
FORMAT=PERCENT6.
FROM CLASS.EMPLOY
WHERE DEPT EQ 'GPO';

Is the same as

DATA MYSET;
SET CLASS.EMPLOY;
WHERE DEPT EQ 'GPO';
BONUSPR = BONUS / SALARY;

PROC PRINT DATA=MYSET;
FORMAT BONUSPR PERCENT6.;
VAR LASTNAME SALARY BONUS BONUSPR;

Data summarization comparison:

PROC SQL;
SELECT DEPT, AVG(SALARY) AS AVSAL,
SUM(SALARY) AS SUMSAL,
MIN(SALARY) AS MINSAL,
MAX(SALARY) AS MAXSAL
FROM CLASS.EMPLOY
GROUP BY DEPT
ORDER BY AVSAL DESC;

Is the same as

PROC MEANS DATA=CLASS.EMPLOY NWAY;
CLASS DEPT;
VAR SALARY;
OUTPUT OUT=MYSET MEAN=AVSAL
SUM=SUMSAL
MIN=MINSAL MAX=MAXSAL;

PROC SORT DATA=MYSET;
BY AVSAL;

PROC PRINT DATA=MYSET NOOBS;
VAR DEPT AVSAL SUMSAL MINSAL MAXSAL;

Conclusion

The SQL procedure is a powerful addition to a SAS programmers information delivery tools. It should not be viewed as a replacement to standard SAS code but as another potential solution.

Some explicit reasons for using the SQL procedure:
1. Merging 3 or more data sets without common keys.
2. Merging on ranges.
3. Summarizing on a calculated value.
4. Calculations involving summary values.
5. Selecting detail rows based on summary values.
6. SQL supports identical column names from different tables.
7. The SQL compiler will figure out the optimum order or index to use for a query.

Some practical reasons for using the SQL procedure:
1. Less code to understand and maintain by a SAS programmer.
2. SQL is ‘standard’ therefore non SAS programmers can ‘read’ SAS programs using the SQL procedure.
3. Task specific SQL code may already exist for another RDBMS that can easily be ported into SAS.

Some reasons not to use the SQL procedure:
1. Does not replace the wide variety of tools available with SAS PROCs.
2. No INFILE, INPUT or FILE, PUT.
3. Only one table (SAS data set) created at a time.
4. Append data is easier than SET operators.
5. Sometimes more difficult to handle matched and unmatched records.

Questions, comments, and suggestions are welcome at:
Chris Yindra
C. Y. Training Associates, Inc
chris@cyassociates.com
www.cyassociates.com

REFERENCES

SAS Institute, Inc, SAS Guide to the SQL Procedure
SAS is a registered trademark or trademark of SAS Institute, Inc., in the US and other countries