# Chapter 1  Overview of the SQL Procedure

1.1 Features of PROC SQL .................................................................................................. 1-3

1.2 Selecting Columns and Rows ....................................................................................... 1-6

1.3 Presenting and Summarizing Data ............................................................................. 1-17

1.4 Joining Tables............................................................................................................... 1-27
1.1 Features of PROC SQL

Objectives

- Understand SQL procedure syntax.

Features of PROC SQL

PROC SQL has the following features:

- is part of Base SAS software
- follows guidelines set by the American National Standards Institute (ANSI)
- can query, combine, create, and update SAS and RDBMS (relational database management system) data
- supports SAS functions such as SUBSTR and MEAN
- supports data set options such as DROP= and RENAME=

You must have the appropriate SAS/ACCESS software to query your RDBMS data. You must have appropriate authority to read, create, or update RDBMS objects.
Features of PROC SQL

- The PROC SQL statement does not need to be repeated with each query.
- Each statement is processed individually.
- No PROC PRINT step is needed to view query results.
- No PROC SORT step is needed to order query results.
- No RUN statement is needed.
- Use a QUIT statement to terminate PROC SQL.

Features of the SELECT Statement

The features of the SELECT statement include the following:

- selects data that meets certain conditions
- groups data
- specifies an order for the data
- formats the data
- queries 1 to 32 tables

Table names and variable names can be 1 to 32 characters in length and are not case-sensitive.

Librefs and filerefs are limited to 8 characters. Starting in SAS®9, format and informat names can be up to 32 characters in length.
SELECT Statement Syntax

General form of the SELECT statement:

```
SELECT column-1<, column-2>...
    FROM table-1|view-1<, table-2|view-2>...
    <WHERE expression>
    <GROUP BY column-1<, column-2>...
    <HAVING expression>
    <ORDER BY column-1<, column-2>... <DESC>>;
QUIT;
```

SELECT specifies the columns to be selected.
FROM specifies the table to be queried.
WHERE subsets the data based on a condition.
GROUP BY classifies the data into groups.
HAVING subsets groups of data based on a group condition.
ORDER BY sorts rows of data by the values of specific columns in ascending order by default.
1.2 Selecting Columns and Rows

Objectives
- Display columns directly from a table.
- Display columns calculated from other columns in a query.
- Eliminate duplicate rows in a query.
- Subset the data displayed in a query.

Retrieving Data from a Table
If you are familiar with a table, you can specify column names to be printed in the SELECT statement.

Example: Print employee IDs, job codes, and salaries.

```
libname airline 'c:\SQL';
proc sql;
    select EmpID, JobCode, Salary
    from airline.payrollmaster;
quit;
```
1.2 Selecting Columns and Rows

Employee IDs, Job Codes, and Salaries

Partial Output

<table>
<thead>
<tr>
<th>Emp ID</th>
<th>Job Code</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1919</td>
<td>TA2</td>
<td>$48,126</td>
</tr>
<tr>
<td>1653</td>
<td>ME2</td>
<td>$49,151</td>
</tr>
<tr>
<td>1400</td>
<td>ME1</td>
<td>$41,677</td>
</tr>
<tr>
<td>1350</td>
<td>FA3</td>
<td>$46,040</td>
</tr>
<tr>
<td>1401</td>
<td>TA3</td>
<td>$54,351</td>
</tr>
<tr>
<td>1499</td>
<td>ME3</td>
<td>$60,235</td>
</tr>
<tr>
<td>1101</td>
<td>SCP</td>
<td>$26,212</td>
</tr>
<tr>
<td>1333</td>
<td>PT2</td>
<td>$124,048</td>
</tr>
<tr>
<td>1402</td>
<td>TA2</td>
<td>$45,661</td>
</tr>
</tbody>
</table>

Retrieving Data from a Table

```sql
proc sql;
select *
from airline.payrollmaster;
quit;
```

Partial Output

<table>
<thead>
<tr>
<th>Emp ID</th>
<th>Gender</th>
<th>Job Code</th>
<th>Salary</th>
<th>DateOfBirth</th>
<th>DateOfHire</th>
</tr>
</thead>
<tbody>
<tr>
<td>1919</td>
<td>M</td>
<td>TA2</td>
<td>$48,126</td>
<td>16SEP1958</td>
<td>07JUN1985</td>
</tr>
<tr>
<td>1653</td>
<td>F</td>
<td>ME2</td>
<td>$49,151</td>
<td>19OCT1962</td>
<td>12AUG1988</td>
</tr>
<tr>
<td>1400</td>
<td>M</td>
<td>ME1</td>
<td>$41,677</td>
<td>08NOV1965</td>
<td>19OCT1988</td>
</tr>
<tr>
<td>1350</td>
<td>F</td>
<td>FA3</td>
<td>$46,040</td>
<td>04SEP1963</td>
<td>01AUG1988</td>
</tr>
<tr>
<td>1401</td>
<td>M</td>
<td>TA3</td>
<td>$54,351</td>
<td>16DEC1948</td>
<td>21NOV1983</td>
</tr>
<tr>
<td>1499</td>
<td>M</td>
<td>ME3</td>
<td>$60,235</td>
<td>29APR1952</td>
<td>11JUN1978</td>
</tr>
</tbody>
</table>

If you are not familiar with a table, an asterisk in the SELECT statement prints all columns in their originally stored order.
Retrieving Data from a Table

```sql
proc sql;
  select * from airline.payrollmaster(DROP=gender);
quit;
```

Partial Output

```
   Emp ID  Job Code  Salary  DateOfBirth  DateOfHire
-------  --------  -------  ------------  -----------
    1919    TA2       $48,126  16SEP1958   07JUN1985
    1653    ME2       $49,151  19OCT1962  12AUG1988
    1400    ME1       $41,677  08NOV1965  19OCT1988
    1350    FA3       $46,040  04SEP1963  01AUG1988
    1401    TA3       $54,351  16DEC1948  21NOV1983
    1499    ME3       $60,235  29APR1952  11JUN1978
```

Expressions

Calculate new columns from existing columns, and name the new columns using the AS keyword.

Example: Calculate employee bonuses and age.

```sql
proc sql;
  select EmpID, JobCode, Salary,
       Salary * .10 as Bonus,
       int((today()-DateOfBirth)/365.25) as Age
  from airline.payrollmaster;
quit;
```
### Employee Bonuses and Age

Partial Output

```
The SAS System

<table>
<thead>
<tr>
<th>Emp ID</th>
<th>Job Code</th>
<th>Salary</th>
<th>Bonus</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1919</td>
<td>TA2</td>
<td>$48,126</td>
<td>4812.64</td>
<td>48</td>
</tr>
<tr>
<td>1653</td>
<td>ME2</td>
<td>$49,151</td>
<td>4915.12</td>
<td>44</td>
</tr>
<tr>
<td>1400</td>
<td>ME1</td>
<td>$41,677</td>
<td>4167.66</td>
<td>41</td>
</tr>
<tr>
<td>1350</td>
<td>FA3</td>
<td>$46,040</td>
<td>4604.04</td>
<td>43</td>
</tr>
<tr>
<td>1401</td>
<td>TA3</td>
<td>$54,361</td>
<td>5435.08</td>
<td>58</td>
</tr>
<tr>
<td>1499</td>
<td>ME3</td>
<td>$60,235</td>
<td>6023.5</td>
<td>55</td>
</tr>
<tr>
<td>1101</td>
<td>SCP</td>
<td>$26,212</td>
<td>2621.22</td>
<td>47</td>
</tr>
<tr>
<td>1300</td>
<td>PT2</td>
<td>$124,048</td>
<td>12404.84</td>
<td>48</td>
</tr>
<tr>
<td>1402</td>
<td>TA2</td>
<td>$45,661</td>
<td>4566.1</td>
<td>46</td>
</tr>
<tr>
<td>1479</td>
<td>TA3</td>
<td>$54,299</td>
<td>5429.9</td>
<td>40</td>
</tr>
</tbody>
</table>
```

### Eliminating Duplicate Rows

Use the DISTINCT keyword to eliminate duplicate rows in query results.

Example: Determine the international flights that were flown during the month.

```
proc sql;
    select distinct FlightNumber, Destination
    from airline.internationalflights;
quit;
```
Eliminating Duplicate Rows

Output

The SAS System

<table>
<thead>
<tr>
<th>FlightNumber</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>132</td>
<td>YYZ</td>
</tr>
<tr>
<td>182</td>
<td>YYZ</td>
</tr>
<tr>
<td>219</td>
<td>LHR</td>
</tr>
<tr>
<td>271</td>
<td>CDG</td>
</tr>
<tr>
<td>387</td>
<td>CPH</td>
</tr>
<tr>
<td>622</td>
<td>FRA</td>
</tr>
<tr>
<td>821</td>
<td>LHR</td>
</tr>
</tbody>
</table>

Subsetting with the WHERE Clause

Use a WHERE clause to specify a condition that the data must satisfy before being selected.

Example: Display all employees that earn more than $112,000.

```
proc sql;
  select EmpID, JobCode, Salary
  from airline.payrollmaster
  where Salary > 112000;
quit;
```
Subsetting with the WHERE Clause

Output

<table>
<thead>
<tr>
<th>Emp ID</th>
<th>Job Code</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1333</td>
<td>PT2</td>
<td>$124,048</td>
</tr>
<tr>
<td>1404</td>
<td>PT2</td>
<td>$127,926</td>
</tr>
<tr>
<td>1118</td>
<td>PT3</td>
<td>$155,931</td>
</tr>
<tr>
<td>1410</td>
<td>PT2</td>
<td>$118,559</td>
</tr>
<tr>
<td>1777</td>
<td>PT3</td>
<td>$153,482</td>
</tr>
<tr>
<td>1106</td>
<td>PT2</td>
<td>$125,485</td>
</tr>
<tr>
<td>1442</td>
<td>PT2</td>
<td>$118,350</td>
</tr>
<tr>
<td>1478</td>
<td>PT2</td>
<td>$117,884</td>
</tr>
<tr>
<td>1890</td>
<td>PT2</td>
<td>$120,254</td>
</tr>
<tr>
<td>1107</td>
<td>PT2</td>
<td>$125,968</td>
</tr>
<tr>
<td>1830</td>
<td>PT2</td>
<td>$118,259</td>
</tr>
<tr>
<td>1928</td>
<td>PT2</td>
<td>$125,801</td>
</tr>
</tbody>
</table>

Subsetting with the WHERE Clause

You can use all common comparison operators in a WHERE clause.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>GT</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>EQ</td>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>LE</td>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>GE</td>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>NE</td>
<td>^=</td>
<td>Not equal to (EBCDIC)</td>
</tr>
<tr>
<td></td>
<td>^=</td>
<td>Not equal to (ASCII)</td>
</tr>
</tbody>
</table>
Subsetting with the WHERE Clause

You can use the IN operator to compare a value to a list of values. If the value matches at least one in the list, the expression is true; otherwise, the expression is false.

```sql
where JobCategory in ('PT', 'NA', 'FA')

where DayOfWeek in (2, 4, 6)
```

Subsetting with the WHERE Clause

You can specify multiple expressions in a WHERE clause by using logical operators.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td></td>
<td>or, either</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>and, both</td>
</tr>
<tr>
<td>NOT</td>
<td>¬</td>
<td>not, negation EBCDIC</td>
</tr>
<tr>
<td>NOT</td>
<td>^</td>
<td>not, negation ASCII</td>
</tr>
</tbody>
</table>
Subsetting with the WHERE Clause

Use either CONTAINS or ? to select rows that include the substring specified.

```sql
where word ? 'LAM'
```

(BLAME, LAMENT, and BEDLAM are selected.)

Use either IS NULL or IS MISSING to select rows with missing values.

```sql
where FlightNumber is missing
```

Subsetting with the WHERE Clause

Use BETWEEN-AND to select rows containing ranges of values, inclusively.

```sql
where Date between '01mar2000'd and '07mar2000'd
```

```sql
where Salary between 70000 and 80000
```
Subsetting with the WHERE Clause

Use **LIKE** to select rows by comparing character values to specified patterns.

A `%` sign replaces any number of characters.

```
where LastName like 'H%'
```

(H plus any characters; for example, HENRY and HAMM.)

A single underscore (`_`) replaces individual characters.

```
where JobCode like '__1'
```

(Captures any two characters and 1, for example, 'FA1'.)

Subsetting with Calculated Values

Example: Display only the flights where the total number of passengers was fewer than 100 people.

```
proc sql;
    select FlightNumber,Date,Destination, Boarded + Transferred + Nonrevenue as Total
    from airline.marchflights
    where Total < 100;
```

Partial Log

```
ERROR: The following columns were not found in the contributing tables: Total.
```
Subsetting with Calculated Values

One solution is to repeat the calculation in the WHERE clause.

```sql
proc sql;
    select FlightNumber, Date, Destination, Boarded+Transferred+Nonrevenue as Total
    from airline.marchflights
    where Boarded+Transferred+Nonrevenue < 100;
```

Subsetting with Calculated Values

A more efficient method is to use the CALCULATED keyword to refer to already calculated columns in the SELECT clause.

```sql
proc sql;
    select FlightNumber, Date, Destination, Boarded + Transferred + Nonrevenue as Total
    from airline.marchflights
    where calculated Total < 100;
```
## Subsetting with Calculated Values

Partial Output

<table>
<thead>
<tr>
<th>FlightNumber</th>
<th>Date</th>
<th>Destination</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>982</td>
<td>01MAR2000</td>
<td>DFW</td>
<td>70</td>
</tr>
<tr>
<td>416</td>
<td>01MAR2000</td>
<td>WAS</td>
<td>93</td>
</tr>
<tr>
<td>829</td>
<td>01MAR2000</td>
<td>WAS</td>
<td>96</td>
</tr>
<tr>
<td>416</td>
<td>02MAR2000</td>
<td>WAS</td>
<td>90</td>
</tr>
<tr>
<td>302</td>
<td>02MAR2000</td>
<td>WAS</td>
<td>93</td>
</tr>
</tbody>
</table>
1.3 Presenting and Summarizing Data

Objectives

- Order the data displayed in a query.
- Use SAS formats, labels, and titles to enhance query output.
- Use functions to summarize data in a query.
- Group data for aggregate functions.

Ordering Data

The ORDER BY clause has the following functionality:

- sorts on any column or expression (display or nondisplay) in ascending order by default
- can sort data in descending order by following the column name with the DESC keyword
- uses a column name or a number that represents the position of an item in the SELECT list
- can sort on multiple columns
Ordering Data

```
proc sql;
  select EmpID, JobCode, Salary
  from airline.payrollmaster
  where JobCode contains 'NA'
  order by Salary desc;
```

Output

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Job Code</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1352</td>
<td>NA2</td>
<td>$75,317</td>
</tr>
<tr>
<td>1417</td>
<td>NA2</td>
<td>$73,178</td>
</tr>
<tr>
<td>1935</td>
<td>NA2</td>
<td>$71,513</td>
</tr>
<tr>
<td>1839</td>
<td>NA1</td>
<td>$60,806</td>
</tr>
<tr>
<td>1443</td>
<td>NA1</td>
<td>$59,184</td>
</tr>
<tr>
<td>1332</td>
<td>NA1</td>
<td>$59,049</td>
</tr>
<tr>
<td>1269</td>
<td>NA1</td>
<td>$58,366</td>
</tr>
<tr>
<td>1111</td>
<td>NA1</td>
<td>$56,820</td>
</tr>
</tbody>
</table>
Enhancing Query Output
You can use SAS titles and footnotes enhance your output.

You can also use SAS formats and labels to customize PROC SQL output. After the column name in the SELECT list, you specify the following:

- LABEL= option to alter the column heading
- FORMAT= option to alter the appearance of the values in that column.

Example: Enhance the report. Display the navigators and their salaries.

```
proc sql;
  title 'Navigator Salaries';
  select EmpID label='Employee Identifier',
          JobCode label='Job Code',
          Salary label='Annual Salary'
    format=dollar12.2
  from airline.payrollmaster
  where JobCode contains 'NA'
  order by Salary desc;
```

Enhanced Query Output

Output

<table>
<thead>
<tr>
<th>Employee Identifier</th>
<th>Job Code</th>
<th>Annual Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1352</td>
<td>NA2</td>
<td>$75,317.20</td>
</tr>
<tr>
<td>1417</td>
<td>NA2</td>
<td>$73,178.00</td>
</tr>
<tr>
<td>1935</td>
<td>NA2</td>
<td>$71,513.40</td>
</tr>
<tr>
<td>1839</td>
<td>NA1</td>
<td>$60,806.20</td>
</tr>
<tr>
<td>1443</td>
<td>NA1</td>
<td>$59,183.60</td>
</tr>
<tr>
<td>1332</td>
<td>NA1</td>
<td>$59,049.20</td>
</tr>
<tr>
<td>1269</td>
<td>NA1</td>
<td>$58,366.00</td>
</tr>
<tr>
<td>1111</td>
<td>NA1</td>
<td>$56,820.40</td>
</tr>
</tbody>
</table>

Summary Functions

Example: Find the total number of passengers for each flight in March.

```
proc sql;
  select Date, FlightNumber, Boarded, Transferred, Nonrevenue,
       sum(Boarded, Transferred, Nonrevenue) as Total
  from airline.marchflights;
quit;
```

This calculation is performed across columns for each row.
Summary Functions

Partial Output

<table>
<thead>
<tr>
<th>Date</th>
<th>FlightNumber</th>
<th>Boarded</th>
<th>Transferred</th>
<th>Nonrevenue</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>30MAR2000</td>
<td>183</td>
<td>86</td>
<td>5</td>
<td>4</td>
<td>95</td>
</tr>
<tr>
<td>30MAR2000</td>
<td>271</td>
<td>164</td>
<td>15</td>
<td>6</td>
<td>185</td>
</tr>
<tr>
<td>30MAR2000</td>
<td>921</td>
<td>101</td>
<td>20</td>
<td>4</td>
<td>125</td>
</tr>
<tr>
<td>30MAR2000</td>
<td>302</td>
<td>54</td>
<td>14</td>
<td>3</td>
<td>71</td>
</tr>
<tr>
<td>30MAR2000</td>
<td>431</td>
<td>159</td>
<td>14</td>
<td>4</td>
<td>177</td>
</tr>
<tr>
<td>30MAR2000</td>
<td>308</td>
<td>121</td>
<td>18</td>
<td>6</td>
<td>145</td>
</tr>
<tr>
<td>31MAR2000</td>
<td>182</td>
<td>91</td>
<td>7</td>
<td>6</td>
<td>104</td>
</tr>
<tr>
<td>31MAR2000</td>
<td>114</td>
<td>183</td>
<td>14</td>
<td>4</td>
<td>201</td>
</tr>
<tr>
<td>31MAR2000</td>
<td>202</td>
<td>111</td>
<td>12</td>
<td>3</td>
<td>126</td>
</tr>
</tbody>
</table>

Summary Functions

If you specify only one column name in a summary function, the statistic is calculated down the column.

Example: Determine the average salary for the company.

```sql
proc sql;
    select avg(Salary) as MeanSalary
    from airline.payrollmaster;
```

Output

```
MeanSalary
54079.65
```
### Summary Functions

The following are selected functions:

- **AVG, MEAN**: mean or average value
- **COUNT, FREQ, N**: number of nonmissing values
- **MAX**: largest value
- **MIN**: smallest value
- **NMISS**: number of missing values
- **STD**: standard deviation
- **SUM**: sum of values
- **VAR**: variance

---

**Example**: Add the JobCode column to the summarized query.

```sql
proc sql;
    select JobCode, avg(Salary) as Average
    from airline.payrollmaster;
```
Summary Functions

Partial Output

<table>
<thead>
<tr>
<th>Job Code</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA2</td>
<td>54079.65</td>
</tr>
<tr>
<td>ME2</td>
<td>54079.65</td>
</tr>
<tr>
<td>ME1</td>
<td>54079.65</td>
</tr>
<tr>
<td>FA3</td>
<td>54079.65</td>
</tr>
<tr>
<td>TA3</td>
<td>54079.65</td>
</tr>
<tr>
<td>ME3</td>
<td>54079.65</td>
</tr>
<tr>
<td>SCP</td>
<td>54079.65</td>
</tr>
<tr>
<td>PT2</td>
<td>54079.65</td>
</tr>
<tr>
<td>TA2</td>
<td>54079.65</td>
</tr>
<tr>
<td>TA3</td>
<td>54079.65</td>
</tr>
<tr>
<td>ME1</td>
<td>54079.65</td>
</tr>
</tbody>
</table>

Grouping Data

You can use the GROUP BY clause to

- classify the data into groups based on the values of one or more columns
- calculate statistics for each unique value of the grouping columns.
Grouping Data

Example: Display the average salary and number of employees for each job code.

```sql
proc sql;
   select JobCode, avg(Salary) as Average format=dollar11.2,
       count(*) as Employees
   from airline.payrollmaster
   group by JobCode;
```

Partial Output

<table>
<thead>
<tr>
<th>Job Code</th>
<th>Average</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCK</td>
<td>$36,111.91</td>
<td>9</td>
</tr>
<tr>
<td>FA1</td>
<td>$32,255.11</td>
<td>11</td>
</tr>
<tr>
<td>FA2</td>
<td>$39,181.63</td>
<td>16</td>
</tr>
<tr>
<td>FA3</td>
<td>$46,107.40</td>
<td>7</td>
</tr>
<tr>
<td>ME1</td>
<td>$39,900.35</td>
<td>8</td>
</tr>
<tr>
<td>ME2</td>
<td>$49,807.60</td>
<td>14</td>
</tr>
<tr>
<td>ME3</td>
<td>$59,375.00</td>
<td>7</td>
</tr>
<tr>
<td>NA1</td>
<td>$58,845.08</td>
<td>5</td>
</tr>
<tr>
<td>NA2</td>
<td>$73,336.20</td>
<td>3</td>
</tr>
<tr>
<td>PT1</td>
<td>$95,071.20</td>
<td>8</td>
</tr>
</tbody>
</table>
Selecting Groups of Data with the HAVING Clause

The WHERE clause selects data based on values for individual rows. To select entire groups of data, use the HAVING clause.

Example: Display all job codes with an average salary of more than the company average salary of $54,079.

```
proc sql;
    select JobCode, avg(Salary) as Average
        format=dollar11.2
    from airline.payrollmaster
    group by JobCode
    having avg(salary) > 54079;
```

To avoid using hard-coded constants, use a subquery to return the company average salary. A subquery, or inner query, is a query-expression that is nested as part of another query-expression. The subquery executes first, returning its results to the outer query.

```
select JobCode, avg(Salary) as Average format=dollar11.2
    from airline.payrollmaster
    group by JobCode
    having avg(Salary) > (select avg(Salary)
                           from airline.payrollmaster);
```
Selecting Groups of Data with the HAVING Clause

Output

<table>
<thead>
<tr>
<th>Job Code</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME3</td>
<td>$59,375.00</td>
</tr>
<tr>
<td>NA1</td>
<td>$58,845.08</td>
</tr>
<tr>
<td>NA2</td>
<td>$73,336.20</td>
</tr>
<tr>
<td>PT1</td>
<td>$95,071.20</td>
</tr>
<tr>
<td>PT2</td>
<td>$122,253.60</td>
</tr>
<tr>
<td>PT3</td>
<td>$154,706.30</td>
</tr>
</tbody>
</table>
1.4 Joining Tables

**Objectives**
- Join two tables.
- Use a table alias.
- Create a table from the join query results.

**Combining Data from Multiple Tables**
Joins combine tables horizontally (side by side).

<table>
<thead>
<tr>
<th>Table A</th>
<th>Table B</th>
</tr>
</thead>
</table>

PROC SQL supports both inner and outer joins. We will look at inner joins.
Inner Joins

Inner joins have the following characteristics:
- return only matching rows
- allow a maximum of 32 tables to be joined at the same time

An inner join returns only the subset of rows from the first table that matches rows from the second table. You can specify the columns that you want to be compared for matching values in a WHERE clause. General form of an inner join:

```
SELECT column-1, column-2, ...
FROM table-1, table-2, ...
WHERE join-condition(s)
    <AND other subsetting conditions>
    <other clauses>;
```

Without a WHERE clause, the join will return a Cartesian product of the tables where each row from the first table is combined with every row from the second table.
Inner Joins

Example: Create a table that contains the names, job codes, and ages of all New York employees.

- Employee names are found in the `airline.staffmaster` table.
- Employee job codes and birth dates are found in the `airline.payrollmaster` table.

```
proc sql;
  title 'New York Employees';
  select FirstName, LastName, JobCode,
       int((today() - DateOfBirth)/365.25) as Age
  from airline.payrollmaster as p,
       airline.staffmaster as s
  where p.EmpID = s.EmpID and State = 'NY'
  order by JobCode;
quit;
```

An alias is a table nickname. You can assign an alias to a table by following the table name in the FROM clause with the AS keyword and a nickname for the table. Then use the alias in other clauses of the QUERY statement.
Inner Joins

Partial Output

<table>
<thead>
<tr>
<th>FirstName</th>
<th>LastName</th>
<th>Job Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUSSELL</td>
<td>LONG</td>
<td>BCK</td>
<td>37</td>
</tr>
<tr>
<td>LEVI</td>
<td>GORDON</td>
<td>BCK</td>
<td>50</td>
</tr>
<tr>
<td>JAMES</td>
<td>PEARSON</td>
<td>BCK</td>
<td>49</td>
</tr>
<tr>
<td>NATHAN</td>
<td>JONES</td>
<td>BCK</td>
<td>42</td>
</tr>
<tr>
<td>THOMAS</td>
<td>BURNETTE</td>
<td>BCK</td>
<td>41</td>
</tr>
<tr>
<td>RICHARD</td>
<td>VANDEUSEN</td>
<td>BCK</td>
<td>48</td>
</tr>
<tr>
<td>JOHN</td>
<td>MARKS</td>
<td>BCK</td>
<td>42</td>
</tr>
<tr>
<td>DEBORAH</td>
<td>WOOD</td>
<td>FA1</td>
<td>38</td>
</tr>
<tr>
<td>LESLIE</td>
<td>JONES</td>
<td>FA1</td>
<td>41</td>
</tr>
<tr>
<td>ANNE</td>
<td>PARKER</td>
<td>FA1</td>
<td>44</td>
</tr>
<tr>
<td>DIANA</td>
<td>FIELDS</td>
<td>FA1</td>
<td>51</td>
</tr>
<tr>
<td>CASEY</td>
<td>RICHARDS</td>
<td>FA1</td>
<td>39</td>
</tr>
</tbody>
</table>

Creating a Table From Query Results

The `CREATE TABLE table-name AS` statement stores the results of the query into a table.

```sql
proc sql;
  create table airline.newyork as
  select FirstName, LastName, JobCode,
         int((today()-DateOfBirth)/365.25) as Age
  from airline.payrollmaster as p,
       airline.staffmaster as s
  where p.EmpID=s.EmpID and State='NY'
  order by JobCode;
quit;
```

The CREATE TABLE statement does not generate output.
Creating a Table From Query Results

```sas
proc print data=airline.newyork noobs;
title 'New York Employees';
run;
```

Partial Output

### New York Employees

<table>
<thead>
<tr>
<th>FirstName</th>
<th>LastName</th>
<th>Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUSSELL</td>
<td>LONG</td>
<td>BCK</td>
<td>37</td>
</tr>
<tr>
<td>LEVI</td>
<td>GORDON</td>
<td>BCK</td>
<td>50</td>
</tr>
<tr>
<td>JAMES</td>
<td>PEARSON</td>
<td>BCK</td>
<td>49</td>
</tr>
<tr>
<td>NATHAN</td>
<td>JONES</td>
<td>BCK</td>
<td>42</td>
</tr>
<tr>
<td>THOMAS</td>
<td>BURNETTE</td>
<td>BCK</td>
<td>41</td>
</tr>
<tr>
<td>RICHARD</td>
<td>VANDEUSEN</td>
<td>BCK</td>
<td>48</td>
</tr>
<tr>
<td>JOHN</td>
<td>MARKS</td>
<td>BCK</td>
<td>42</td>
</tr>
<tr>
<td>DEBORAH</td>
<td>WOOD</td>
<td>FA1</td>
<td>38</td>
</tr>
<tr>
<td>LESLIE</td>
<td>JONES</td>
<td>FA1</td>
<td>41</td>
</tr>
</tbody>
</table>

Where to Go Next

SQL Processing with the SAS® System – 2 days

**PROC SQL: Beyond the Basics Using SAS®**
by Kirk Paul Laffler

**The Essential PROC SQL Handbook for SAS® Users**
by Katherine Prairie

SQL Self-Paced eLearning:
http://www.sas.com/apps/elearning/