Overview

Introduction

Sometimes you need quick answers to questions about your data. You might want to query (retrieve data from) a single SAS data set or a combination of data sets to

- examine relationships between data values
- view a subset of your data
- compute values quickly.

The SQL procedure (PROC SQL) provides an easy, flexible way to query and combine your data. This chapter shows you how to create a basic query using one or more tables (data sets). You'll also learn how to create a new table from your query.

Objectives

In this chapter, you learn to

- invoke the SQL procedure
- select columns
- define new columns
- specify the table(s) to be read
- specify subsetting criteria
- order rows by values of one or more columns
- group results by values of one or more columns
- end the SQL procedure
- summarize data
- generate a report as the output of a query
- create a table as the output of a query.

PROC SQL Basics

PROC SQL is the SAS implementation of Structured Query Language (SQL), which is a standardized language that is widely used to retrieve and update data in tables and in views that are based on those tables.
The following chart shows terms used in data processing, SAS, and SQL that are synonymous. The SQL terms are used in this chapter.

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>SAS</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>SAS data set</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>

PROC SQL can often be used as an alternative to other SAS procedures or the DATA step. You can use PROC SQL to
- retrieve data from and manipulate SAS tables
- add or modify data values in a table
- add, modify, or drop columns in a table
- create tables and views
- join multiple tables (whether or not they contain columns with the same name)
- generate reports.

Like other SAS procedures, PROC SQL also enables you to combine data from two or more different types of data sources and present them as a single table. For example, you can combine data from two different types of external databases, or you can combine data from an external database and a SAS data set.

How PROC SQL Is Unique

PROC SQL differs from most other SAS procedures in several ways:

- Unlike other PROC statements, many statements in PROC SQL are composed of clauses. For example, the following PROC SQL step contains two statements: the PROC SQL statement and the SELECT statement. The SELECT statement contains several clauses: SELECT, FROM, WHERE, and ORDER BY.

```sql
proc sql;
  select empid, jobcode, salary,
       salary*.06 as bonus
  from sasuser.payrollmaster
  where salary<32000
  order by jobcode;
```

- The PROC SQL step does not require a RUN statement. PROC SQL executes each query automatically. If you use a RUN statement with a PROC SQL step, SAS
ignores the RUN statement, executes the statements as usual, and generates the note shown below in the SAS log.

Table 1.1 SAS Log

```
1884 proc sql;
1885 select empid,jobcode,salary,
1886 salary*.06 as bonus
1887 from sasuser.payrollmaster
1888 where salary<32000
1889 order by jobcode;
1890 run;
NOTE: PROC SQL statements are executed immediately;
The RUN statement has no effect.
```

Unlike many other SAS procedures, PROC SQL continues to run after you submit a step. To end the procedure, you must submit another PROC step, a DATA step, or a QUIT statement, as shown:

```
proc sql;
  select empid,jobcode,salary, salary*.06 as bonus
  from sasuser.payrollmaster
  where salary<32000
  order by jobcode;
quit;
```

When you submit a PROC SQL step without ending it, the status line displays the message **PROC SQL running**.

**Note:** As a precaution, SAS Enterprise Guide automatically adds a QUIT statement to your code when you submit it to SAS. However, you should get in the habit of adding the QUIT statement to your code. △

**Writing a PROC SQL Step**

Before creating a query, you must first reference the library in which your table is stored. Then you write a PROC SQL step to query your table.
General form, basic PROC SQL step to perform a query:

```
PROC SQL;
    SELECT column-1,<...column-n>
    FROM table-1|view-1,<...table-n|view-n>
    <WHERE expression>
    <GROUP BY column-1,<...column-n>>
    <ORDER BY column-1,<...column-n>>;
```

where

- PROC SQL invokes the SQL procedure
- SELECT specifies the column(s) to be selected
- FROM specifies the table(s) to be queried
- WHERE subsets the data based on a condition
- GROUP BY classifies the data into groups based on the specified column(s)
- ORDER BY sorts the rows that the query returns by the value(s) of the specified column(s).

**CAUTION:**
Unlike other SAS procedures the order of clauses with a SELECT statement in PROC SQL is important. Clauses must appear in the order shown above.

*Note:* A query can also include a HAVING clause, which is introduced at the end of this chapter. To learn more about the HAVING clause, see Chapter 2, “Performing Advanced Queries Using PROC SQL,” on page 25.

### The SELECT Statement

The SELECT statement, which follows the PROC SQL statement, retrieves and displays data. It is composed of clauses, each of which begins with a keyword and is followed by one or more components. The SELECT statement in the following sample code contains four clauses: the required clauses SELECT and FROM, and the optional clauses WHERE and ORDER BY. The end of the statement is indicated by a semicolon.

```
proc sql;
|---select empid,jobcode,salary,
|   salary*.06 as bonus
|---from sasuser.payrollmaster
|---where salary<32000
|---order by jobcode;
```

*Note:* A PROC SQL step that contains one or more SELECT statements is referred to as a PROC SQL query. The SELECT statement is only one of several statements that can be used with PROC SQL.

The following PROC SQL query creates the output report that is shown below:
A PROC SQL query produces a result set that can be output as a report, a table, or a PROC SQL view.

<table>
<thead>
<tr>
<th>Type of Output</th>
<th>PROC SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>report</td>
<td>SELECT</td>
</tr>
<tr>
<td>table</td>
<td>CREATE TABLE and SELECT</td>
</tr>
<tr>
<td>PROC SQL view</td>
<td>CREATE VIEW and SELECT</td>
</tr>
</tbody>
</table>

Note: The CREATE TABLE statement is introduced later in this chapter. You can learn about creating tables in Chapter 5, “Creating and Managing Tables Using PROC SQL,” on page 157. You can learn more about PROC SQL views in Chapter 7, “Creating and Managing Views Using PROC SQL,” on page 241.

You will learn more about the SELECT statement in the following sections.

### Selecting Columns

To specify which column(s) to display in a query, you write a SELECT clause, the first clause in the SELECT statement. After the keyword SELECT, list one or more column names and separate the column names with commas. In the SELECT clause, you can both specify existing columns (columns that are already stored in a table) and create new columns.

The following SELECT clause specifies the columns EmpID, JobCode, Salary, and bonus. The columns EmpID, JobCode, and Salary are existing columns. The column named bonus is a new column.

```sql
proc sql;
    select empid, jobcode, salary,
           salary*.06 as bonus
    from sasuser.payrollmaster
    where salary<32000
    order by jobcode;
```
Creating New Columns

You can create new columns that contain either text or a calculation. New columns will appear in output, along with any existing columns that are selected. Keep in mind that new columns exist only for the duration of the query, unless a table or a view is created.

To create a new column, include any valid SAS expression in the SELECT clause list of columns. You can optionally assign a column alias, a name, to a new column by using the keyword AS followed by the name that you would like to use.

*Note:* A column alias must follow the rules for SAS names.

In the sample PROC SQL query, shown below, an expression is used to calculate the new column: the values of `Salary` are multiplied by .06. The keyword AS is used to assign the column alias `bonus` to the new column.

```sql
proc sql;
  select empid, jobcode, salary,
       salary*.06 as bonus
  from sasuser.payrollmaster
  where salary<32000
  order by jobcode;
```

A column alias is useful because it allows you to reference the column elsewhere in the query.

*Note:* You can learn more about referencing a calculated column from other clauses in Chapter 2, “Performing Advanced Queries Using PROC SQL,” on page 25.

Also, the column alias will appear as a column heading in the output.

The following output shows how the calculated column `bonus` is displayed. Notice that the column alias `bonus` appears in lowercase, exactly as it is specified in the SELECT clause.

<table>
<thead>
<tr>
<th>EmpID</th>
<th>JobCode</th>
<th>Salary</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1370</td>
<td>FA1</td>
<td>$31,681</td>
<td>1999.66</td>
</tr>
<tr>
<td>1422</td>
<td>FA1</td>
<td>$31,438</td>
<td>1886.16</td>
</tr>
<tr>
<td>1113</td>
<td>FA1</td>
<td>$31,314</td>
<td>1878.84</td>
</tr>
<tr>
<td>1132</td>
<td>FA1</td>
<td>$31,378</td>
<td>1882.68</td>
</tr>
<tr>
<td>1094</td>
<td>FA1</td>
<td>$31,175</td>
<td>1870.5</td>
</tr>
<tr>
<td>1789</td>
<td>SCP</td>
<td>$25,656</td>
<td>1539.36</td>
</tr>
<tr>
<td>1584</td>
<td>SCP</td>
<td>$26,398</td>
<td>1581.96</td>
</tr>
<tr>
<td>1354</td>
<td>SCP</td>
<td>$25,669</td>
<td>1540.14</td>
</tr>
<tr>
<td>1101</td>
<td>SCP</td>
<td>$26,212</td>
<td>1572.73</td>
</tr>
<tr>
<td>1658</td>
<td>SCP</td>
<td>$25,120</td>
<td>1507.2</td>
</tr>
<tr>
<td>1405</td>
<td>SCP</td>
<td>$25,278</td>
<td>1516.86</td>
</tr>
<tr>
<td>1104</td>
<td>SCP</td>
<td>$25,124</td>
<td>1507.44</td>
</tr>
</tbody>
</table>

In the SELECT clause, you can optionally specify a label for an existing or a new column. If both a label and a column alias are specified for a new column, the label will be displayed as the column heading in the output. If only a column alias is specified, it is important that you specify the column alias exactly as you want it to appear in the output.
Specifying the Table

After writing the SELECT clause, you specify the table to be queried in the FROM clause. Type the keyword `FROM`, followed by the name of the table, as shown:

```sql
proc sql;
    select empid, jobcode, salary,
        salary*.06 as bonus
    from sasuser.payrollmaster
    where salary<32000
    order by jobcode;
```

The PROC SQL step above queries the permanent SAS table `Payrollmaster`, which is stored in a SAS library to which the libref `Sasuser` has been assigned.

Specifying Subsetting Criteria

To subset data based on a condition, use a WHERE clause in the SELECT statement. As in the WHERE statement and the WHERE command used in other SAS procedures, the expression in the WHERE clause can be any valid SAS expression. In the WHERE clause, you can specify any column(s) from the underlying table(s). The columns specified in the WHERE clause do not have to be specified in the SELECT clause.

In the following PROC SQL query, the WHERE clause selects rows in which the value of the column `Salary` is less than 32,000. The output is also shown.

```sql
proc sql;
    select empid, jobcode, salary,
        salary*.06 as bonus
    from sasuser.payrollmaster
    where salary<32000
    order by jobcode;
```

<table>
<thead>
<tr>
<th>EmpID</th>
<th>JobCode</th>
<th>Salary</th>
<th>Bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1570</td>
<td>FA1</td>
<td>$31,661</td>
<td>1639.66</td>
</tr>
<tr>
<td>1422</td>
<td>FA1</td>
<td>$91,436</td>
<td>1688.16</td>
</tr>
<tr>
<td>1113</td>
<td>FA1</td>
<td>$31,314</td>
<td>1678.84</td>
</tr>
<tr>
<td>1132</td>
<td>FA1</td>
<td>$31,378</td>
<td>1682.68</td>
</tr>
<tr>
<td>1094</td>
<td>FA1</td>
<td>$31,175</td>
<td>1670.5</td>
</tr>
<tr>
<td>1799</td>
<td>SCP</td>
<td>$25,666</td>
<td>1638.36</td>
</tr>
<tr>
<td>1964</td>
<td>SCP</td>
<td>$35,368</td>
<td>1851.98</td>
</tr>
<tr>
<td>1354</td>
<td>SCP</td>
<td>$25,069</td>
<td>1640.14</td>
</tr>
<tr>
<td>1101</td>
<td>SCP</td>
<td>$35,212</td>
<td>1872.72</td>
</tr>
<tr>
<td>1692</td>
<td>SCP</td>
<td>$25,120</td>
<td>1607.2</td>
</tr>
<tr>
<td>1405</td>
<td>SCP</td>
<td>$25,278</td>
<td>1616.89</td>
</tr>
<tr>
<td>1104</td>
<td>SCP</td>
<td>$25,124</td>
<td>1607.44</td>
</tr>
</tbody>
</table>

Ordering Rows

The order of rows in the output of a PROC SQL query cannot be guaranteed, unless you specify a sort order. To sort rows by the values of specific columns, you can use the
ORDER BY clause in the SELECT statement. Specify the keywords ORDER BY, followed by one or more column names separated by commas.

In the following PROC SQL query, the ORDER BY clause sorts rows by values of the column **JobCode**:

```sql
proc sql;
  select empid, jobcode, salary,
    salary*.06 as bonus
  from sasuser.payrollmaster
  where salary<32000
  order by jobcode;
```

**Note:** In this example, the ORDER BY clause is the last clause in the SELECT statement, so the ORDER BY clause ends with a semicolon.

In the output of the sample query, shown below, the rows are sorted by the values of **JobCode**. By default, the ORDER BY clause sorts rows in *ascending* order.

<table>
<thead>
<tr>
<th>EmpID</th>
<th>JobCode</th>
<th>Salary</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td>FA1</td>
<td>$31,661</td>
<td>1898.66</td>
</tr>
<tr>
<td>1422</td>
<td>FA1</td>
<td>$31,436</td>
<td>1886.16</td>
</tr>
<tr>
<td>1113</td>
<td>FA1</td>
<td>$31,314</td>
<td>1878.84</td>
</tr>
<tr>
<td>1132</td>
<td>FA1</td>
<td>$31,370</td>
<td>1882.69</td>
</tr>
<tr>
<td>1094</td>
<td>FA1</td>
<td>$31,175</td>
<td>1870.55</td>
</tr>
<tr>
<td>1789</td>
<td>SCP</td>
<td>$25,656</td>
<td>1539.35</td>
</tr>
<tr>
<td>1564</td>
<td>SCP</td>
<td>$25,366</td>
<td>1581.96</td>
</tr>
<tr>
<td>1354</td>
<td>SCP</td>
<td>$25,669</td>
<td>1540.14</td>
</tr>
<tr>
<td>1101</td>
<td>SCP</td>
<td>$25,212</td>
<td>1572.72</td>
</tr>
<tr>
<td>1658</td>
<td>SCP</td>
<td>$25,120</td>
<td>1507.2</td>
</tr>
<tr>
<td>1405</td>
<td>SCP</td>
<td>$25,278</td>
<td>1516.68</td>
</tr>
<tr>
<td>1104</td>
<td>SCP</td>
<td>$25,124</td>
<td>1507.44</td>
</tr>
</tbody>
</table>

To sort rows in *descending* order, specify the keyword DESC following the column name. For example, the preceding ORDER BY clause could be modified as follows:

```sql
order by jobcode desc;
```

In the ORDER BY clause, you can alternatively reference a column by its *position* in the SELECT clause list rather than by name. Use an integer to indicate the column's position. The ORDER BY clause in the preceding PROC SQL query has been modified, below, to specify the column **JobCode** by its position in the SELECT clause list (2) rather than by name:

```sql
proc sql;
  select empid, jobcode, salary,
    salary*.06 as bonus
  from sasuser.payrollmaster
  where salary<32000
  order by 2;
```
Ordering by Multiple Columns

To sort rows by the values of two or more columns, list multiple column names (or numbers) in the ORDER BY clause, and use commas to separate the column names (or numbers). In the following PROC SQL query, the ORDER BY clause sorts by the values of two columns, JobCode and EmpID:

```
proc sql;
  select empid,jobcode,salary,
       salary*.06 as bonus
  from sasuser.payrollmaster
  where salary<32000
  order by jobcode,empid;
```

The rows are sorted first by JobCode and then by EmpID, as shown in the following output.

<table>
<thead>
<tr>
<th>EmpID</th>
<th>JobCode</th>
<th>Salary</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1394</td>
<td>FA1</td>
<td>$31,175</td>
<td>1870.5</td>
</tr>
<tr>
<td>1113</td>
<td>FA1</td>
<td>$31,314</td>
<td>1878.64</td>
</tr>
<tr>
<td>1132</td>
<td>FA1</td>
<td>$31,378</td>
<td>1882.68</td>
</tr>
<tr>
<td>1422</td>
<td>FA1</td>
<td>$31,436</td>
<td>1896.16</td>
</tr>
<tr>
<td>1970</td>
<td>FA1</td>
<td>$31,661</td>
<td>1899.66</td>
</tr>
<tr>
<td>1101</td>
<td>SCP</td>
<td>$25,212</td>
<td>1572.72</td>
</tr>
<tr>
<td>1104</td>
<td>SCP</td>
<td>$25,124</td>
<td>1507.44</td>
</tr>
<tr>
<td>1354</td>
<td>SCP</td>
<td>$25,659</td>
<td>1540.14</td>
</tr>
<tr>
<td>1405</td>
<td>SCP</td>
<td>$25,278</td>
<td>1516.68</td>
</tr>
<tr>
<td>1564</td>
<td>SCP</td>
<td>$25,366</td>
<td>1581.96</td>
</tr>
<tr>
<td>1658</td>
<td>SCP</td>
<td>$25,120</td>
<td>1507.2</td>
</tr>
<tr>
<td>1769</td>
<td>SCP</td>
<td>$25,656</td>
<td>1539.36</td>
</tr>
</tbody>
</table>

Note: You can mix the two types of column references, names and numbers, in the ORDER BY clause. For example, the preceding ORDER BY clause could be rewritten as follows:

```
order by 2,empid;
```

Querying Multiple Tables

This topic deals with the more complex task of extracting data from two or more tables.

Previously, you learned how to write a PROC SQL step to query a single table. Suppose you now want to examine data that is stored in two tables. PROC SQL allows you to combine tables horizontally, in other words, to combine rows of data.
In SQL terminology, combining tables horizontally is called *joining* tables. Joins do not alter the original tables.

Suppose you want to create a report that displays the following information for employees of a company: employee identification number, last name, original salary, and new salary. There is no single table that contains all of these columns, so you will have to join the two tables `Sasuser.Salcomps` and `Sasuser.Newsals`. In your query, you want to select four columns, two from the first table and two from the second table. You also need to be sure that the rows you join belong to the same employee. To check this, you want to match employee identification numbers for rows that you merge and to select only the rows that match.

This type of join is known as an *inner join*. An inner join returns a result set for all of the rows in a table that have one or more matching rows in another table.

Note: For more information about PROC SQL joins, see Chapter 3, “Combining Tables Horizontally Using PROC SQL,” on page 79.

Now let’s see how you write a PROC SQL step to combine tables. To join two tables for a query, you can use a PROC SQL step such as the one below. This step uses the SELECT statement to join data from the tables `Salcomps` and `Newsals`. Both of these tables are stored in a SAS library to which the libref `Sasuser` has been assigned.

```sql
proc sql;
    select salcomps.empid, lastname,
          newsals.salary, newsalary
    from sasuser.salcomps, sasuser.newsals
    where salcomps.empid=newsals.empid
    order by lastname;
```

Let’s take a closer look at each clause of this PROC SQL step.

### Specifying Columns That Appear in Multiple Tables

When you join two or more tables, list the columns that you want to select from both tables in the SELECT clause. Separate all column names with commas.

If the tables that you are querying contain same-named columns and you want to list one of these columns in the SELECT clause, you must specify a table name as a prefix for that column.

Note: Prefixing a table name to a column name is called *qualifying* the column name.

The following PROC SQL step joins the two tables `Sasuser.Salcomps` and `Sasuser.Newsals`, both of which contain columns named `EmpID` and `Salary`. To tell PROC SQL where to read the columns `EmpID` and `Salary`, the SELECT clause specifies the table name `Salcomps` as a prefix for `Empid`, and `Newsals` as a prefix for `Salary`. 
Specifying Multiple Table Names

When you join multiple tables in a PROC SQL query, you specify each table name in the FROM clause, as shown below:

```sql
proc sql;
    select salcomps.empid, lastname,
            newsals.salary, newsalary
    from sasuser.salcomps, sasuser.newsals
    where salcomps.empid = newsals.empid
    order by lastname;
```

As in the SELECT clause, you separate names in the FROM clause (in this case, table names) with commas.

Subsetting Rows

As in a query on a single table, the WHERE clause in the SELECT statement selects rows from two or more tables, based on a condition. When you join multiple tables, be sure that the WHERE clause specifies columns with data whose values match, to avoid unwanted combinations.

In the following example, the WHERE clause selects only rows in which the value for EmpID in Sasuser.Salcomps matches the value for EmpID in Sasuser.Newsals. Qualified column names must be used in the WHERE clause to specify each of the two EmpID columns.

```sql
proc sql;
    select salcomps.empid, lastname,
            newsals.salary, newsalary
    from sasuser.salcomps, sasuser.newsals
    where salcomps.empid = newsals.empid
    order by lastname;
```

The output is shown, in part, below.
Performing Queries Using PROC SQL

Ordering Rows

As in PROC SQL steps that query just one table, the ORDER BY clause specifies which column(s) should be used to sort rows in the output. In the following query, the rows will be sorted by LastName:

```sql
proc sql;
select salcomps.empid,lastname, newsals.salary,newsalary
from sasuser.salcomps,sasuser.newsals
where salcomps.empid=newsals.empid
order by lastname;
```

Note: In the table Sasuser.Newsals, the Salary column has the label Employee Salary, as shown in this output.

Caution:
If you join tables that don't contain one or more columns with matching data values, you can produce a huge amount of output.
Summarizing Groups of Data

So far you’ve seen PROC SQL steps that create detail reports. But you might also want to summarize data in groups. To group data for summarizing, you can use the GROUP BY clause. The GROUP BY clause is used in queries that include one or more summary functions. Summary functions produce a statistical summary for each group that is defined in the GROUP BY clause.

Let’s look at the GROUP BY clause and summary functions more closely.

Example

Suppose you want to determine the total number of miles traveled by frequent-flyer program members in each of three membership classes (Gold, Silver, and Bronze). Frequent-flyer program information is stored in the table Sasuser.Frequentflyers. To summarize your data, you can submit the following PROC SQL step:

```
proc sql;
  select membertype
     , sum(milestraveled) as TotalMiles
  from sasuser.frequentflyers
  group by membertype;
```

In this case, the SUM function totals the values of the MilesTraveled column to create the TotalMiles column. The GROUP BY clause groups the data by the values of MemberType.

As in the ORDER BY clause, in the GROUP BY clause you specify the keywords GROUP BY, followed by one or more column names separated by commas.

The results show total miles by membership class (MemberType).

<table>
<thead>
<tr>
<th>MemberType</th>
<th>TotalMiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRONZE</td>
<td>3223225</td>
</tr>
<tr>
<td>GOLD</td>
<td>2903569</td>
</tr>
<tr>
<td>SILVER</td>
<td>4345169</td>
</tr>
</tbody>
</table>

Note: If you specify a GROUP BY clause in a query that does not contain a summary function, your clause is changed to an ORDER BY clause, and a message to that effect is written to the SAS log.

Summary Functions

To summarize data, you can use the following summary functions with PROC SQL. Notice that some functions have more than one name to accommodate both SAS and SQL conventions. Where multiple names are listed, the first name is the SQL name.

- AVG, MEAN: mean or average of values
- COUNT, FREQ, N: number of nonmissing values
- CSS: corrected sum of squares
- CV: coefficient of variation (percent)
- MAX: largest value
Performing Queries Using PROC SQL

Example

Creating Output Tables

To create a new table from the results of a query, use a `CREATE TABLE` statement that includes the keyword `AS` and the clauses that are used in a PROC SQL query: `SELECT`, `FROM`, and any optional clauses, such as `ORDER BY`. The `CREATE TABLE` statement stores your query results in a table instead of displaying the results as a report.

General form, basic PROC SQL step for creating a table from a query result:

```
PROC SQL;
CREATE TABLE table-name AS
SELECT column-1,...column-n
FROM table-1|view-1,...table-n|view-n
<WHERE expression>
<GROUP BY column-1,... column-n>
<ORDER BY column-1,... column-n>;
```

where

- `table-name` specifies the name of the table to be created.

Note: A query can also include a `HAVING` clause, which is introduced at the end of this chapter. To learn more about the `HAVING` clause, see Chapter 2, “Performing Advanced Queries Using PROC SQL,” on page 25.

Example

Suppose that after determining the total miles traveled for each frequent-flyer membership class in the `Sasuser.Frequentflyers` table, you want to store this information in the temporary table `Work.Miles`. To do so, you can submit the following PROC SQL step:
proc sql;
  create table work.miles as
    select membertype
    sum(milestraveled) as TotalMiles
  from sasuser.frequentflyers
  group by membertype;

Because the CREATE TABLE statement is used, this query does not create a report. The SAS log verifies that the table was created and indicates how many rows and columns the table contains.

Table 1.2  SAS Log

| NOTE: Table WORK.MILES created, with 3 rows and 2 columns. |

Tip: In this example, you are instructed to save the data to a temporary table that will be deleted at the end of the SAS session. To save the table permanently in the Sasuser library, use the libref Sasuser instead of the libref Work in the CREATE TABLE clause.

Additional Features

To further refine a PROC SQL query that contains a GROUP BY clause, you can use a HAVING clause. A HAVING clause works with the GROUP BY clause to restrict the groups that are displayed in the output, based on one or more specified conditions.

For example, the following PROC SQL query groups the output rows by JobCode. The HAVING clause uses the summary function AVG to specify that only the groups that have an average salary that is greater than 40,000 will be displayed in the output.

proc sql;
  select jobcode,avg(salary) as Avg
  from sasuser.payrollmaster
  group by jobcode
  having avg(salary)>40000
  order by jobcode;

Note: You can learn more about the use of the HAVING clause in Chapter 2, “Performing Advanced Queries Using PROC SQL,” on page 25. △
This section contains the following:
- a text summary of the material taught in this chapter
- syntax for statements and options
- sample programs
- points to remember.

Text Summary

PROC SQL Basics

PROC SQL uses statements that are written in Structured Query Language (SQL), which is a standardized language that is widely used to retrieve and update data in tables and in views that are based on those tables. When you want to examine relationships between data values, subset your data, or compute values, the SQL procedure provides an easy, flexible way to analyze your data.

PROC SQL differs from most other SAS procedures in several ways:
- Many statements in PROC SQL, such as the SELECT statement, are composed of clauses.
- The PROC SQL step does not require a RUN statement.
- PROC SQL continues to run after you submit a step. To end the procedure, you must submit another PROC step, a DATA step, or a QUIT statement.

Writing a PROC SQL Step

Before creating a query, you must assign a libref to the SAS data library in which the table to be used is stored. Then you submit a PROC SQL step. You use the PROC SQL statement to invoke the SQL procedure.

Selecting Columns

To specify which column(s) to display in a query, you write a SELECT clause as the first clause in the SELECT statement. In the SELECT clause, you can specify existing columns and create new columns that contain either text or a calculation.

Specifying the Table

You specify the table to be queried in the FROM clause.

Specifying Subsetting Criteria

To subset data based on a condition, write a WHERE clause that contains an expression.

Ordering Rows

The order of rows in the output of a PROC SQL query cannot be guaranteed, unless you specify a sort order. To sort rows by the values of specific columns, use the ORDER BY clause.
Querying Multiple Tables

You can use a PROC SQL step to query data that is stored in two or more tables. In SQL terminology, this is called *joining* tables. Follow these steps to join multiple tables:

1. Specify column names from one or both tables in the SELECT clause and, if you are selecting a column that has the same name in multiple tables, prefix the table name to that column name.
2. Specify each table name in the FROM clause.
3. Use the WHERE clause to select rows from two or more tables, based on a condition.
4. Use the ORDER BY clause to sort rows that are retrieved from two or more tables by the values of the selected column(s).

Summarizing Groups of Data

You can use a GROUP BY clause in your PROC SQL step to summarize data in groups. The GROUP BY clause is used in queries that include one or more summary functions. Summary functions produce a statistical summary for each group that is defined in the GROUP BY clause.

Creating Output Tables

To create a new table from the results of your query, you can use the CREATE TABLE statement in your PROC SQL step. This statement enables you to store your results in a table instead of displaying the query results as a report.

Additional Features

To further refine a PROC SQL query that contains a GROUP BY clause, you can use a HAVING clause. A HAVING clause works with the GROUP BY clause to restrict the groups that are displayed in the output, based on one or more specified conditions.

Syntax

```
LIBNAME libref 'SAS-data-library';
PROC SQL;
CREATE TABLE table-name AS
SELECT column-1,...,column-n
FROM table-1|view-1,...,table-n|view-n
WHERE expression
GROUP BY column-1,...,column-n
ORDER BY column-1,...,column-n;
QUIT;
```

Sample Programs

Querying a Table

```
proc sql;
select empid,jobcode,salary,
salary*.06 as bonus
```
Performing Queries Using PROC SQL

from sasuser.payrollmaster
where salary<32000
order by jobcode;
quit;

Summarizing Groups of Data

proc sql;
  select membertype,
       sum(milestraveled) as TotalMiles
  from sasuser.frequentflyers
  group by membertype;
quit;

Creating a Table from the Results of a Query on Two Tables

proc sql;
  create table work.miles as
    select salcomps.empid, lastname,
           newsals.salary.newsalary
    from sasuser.salcomps.sasuser.newsals
    where salcomps.empid=newsals.empid
    order by 2;
quit;

Points to Remember

- Do not use a RUN statement with the SQL procedure.
- Do not end a clause with a semicolon unless it is the last clause in the statement.
- When you join multiple tables, be sure to specify columns that have matching data values in the WHERE clause in order to avoid unwanted combinations.
- To end the SQL procedure, you can submit another PROC step, a DATA step, or a QUIT statement.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which of the clauses in the PROC SQL program below is written incorrectly?

   proc sql;
   select style sqfeet bedrooms
       from choice.houses
       where sqfeet ge 800;

   a SELECT
   b FROM
   c WHERE
   d both a and c

2. How many statements does the program below contain?

   proc sql;
   select grapes, oranges,
grapes + oranges as sumsales
        from sales.produce
        order by sumsales;

a  two
b  three
c  four
d  five

3 Complete the following PROC SQL query to select the columns Address and SqFeet from the table List.Size and to select Price from the table List.Price. (Only the Address column appears in both tables.)

```sql
proc sql;
    select address,sqfeet,price
    from list.size,list.price;
```

a  select address,sqfeet,price
b  select size.address,sqfeet,price
c  select price.address,sqfeet,price
d  either b or c

4 Which of the clauses below correctly sorts rows by the values of the columns Price and SqFeet?

a  order price, sqfeet
b  order by price,sqfeet
c  sort by price sqfeet
d  sort price sqfeet

5 Which clause below specifies that the two tables Produce and Hardware be queried? Both tables are located in a library to which the libref Sales has been assigned.

a  select sales.produce sales.hardware
b  from sales.produce sales.hardware
c  from sales.produce, sales.hardware
d  where sales.produce, sales.hardware

6 Complete the SELECT clause below to create a new column named Profit by subtracting the values of the column Cost from those of the column Price.

```sql
select fruit,cost,price,
    __________________
```

a  Profit=price-cost
b  price-cost as Profit
c  profit=price-cost
d  Profit as price-cost
7. What happens if you use a GROUP BY clause in a PROC SQL step without a summary function?
   a. The step does not execute.
   b. The first numeric column is summed by default.
   c. The GROUP BY clause is changed to an ORDER BY clause.
   d. The step executes but does not group or sort data.

8. If you specify a CREATE TABLE statement in your PROC SQL step,
   a. the results of the query are displayed, and a new table is created.
   b. a new table is created, but it does not contain any summarization that was specified in the PROC SQL step.
   c. a new table is created, but no report is displayed.
   d. results are grouped by the value of the summarized column.

9. Which statement is true regarding the use of the PROC SQL step to query data that is stored in two or more tables?
   a. When you join multiple tables, the tables must contain a common column.
   b. You must specify the table from which you want each column to be read.
   c. The tables that are being joined must be from the same type of data source.
   d. If two tables that are being joined contain a same-named column, then you must specify the table from which you want the column to be read.

10. Which clause in the following program is incorrect?

    ```sql
    proc sql;
    select sex, mean(weight) as avgweight
    from company.employees company.health
    where employees.id=health.id
    group by sex;
    ```

    a. SELECT
    b. FROM
    c. WHERE
    d. GROUP BY