

SQL Basics

Prepared by



Global Business Intelligence Consulting and Training

Destiny Corporation
100 Great Meadow Rd Suite 601
Wethersfield, CT 06109-2379
Phone: (860) 721-1684 1-800-7TRAINING Fax:
(860) 721-9784
Email: info@destinycorp.com
Web: www.destinycorp.com

Objectives of Proc SQL

The basic ideas behind the design of the language are:

- To provide a simple way to extract data from other tables of data
- To allow the user to concentrate on the data to be extracted rather than worry how the data is to be obtained
- To by-pass the need to specify the physical attributes about the data, such as its structure, location and type
- To be able to change the structure of the data without having to change the application that uses the data.

Concepts

The Query

The heart of SQL is the Query.

The Query is the ability to define a set of data to be extracted and computed from one or more databases or tables of data.

Terminology

The following table compares SQL terminology across three environments.

SQL-Type Terminology	SAS Terminology	DP Terminology
Table	Data Set	File
Row	Observation	Record
Column	Variable	Field

As this table illustrates, an SQL table consists of a series of rows and columns comparable to observations and variables in a SAS data set. In fact, the two structures can be viewed as being identical.

There is an important difference between the two.

- A SAS data file contains the concept of a sequence of ordered observations i.e. you can extract observations 10 to 50, or use structures like the First.By variable.
- An SQL table has no inherent concept of the ordering of its rows.

A New Way of Thinking

Traditionally, SAS data sets are processed sequentially within the Data step.

```
Command window
00001 data new;
00002 set old;
00003 if id = 'A';
00004 run;
00005 proc print data=new;
00006 run;
```

Each observation is read from the input data set (old) and written to the Output data set (new).

In Proc SQL the comparable code would read as follows.

```
Command window
00001 proc sql;
00002 create table new as
00003 select *
00004 from old
00005 where type = 'A';
00006 quit;
```

There is no concept of a sequential read of the input data table.

The selection is done Set-Wise. Physically, the required rows are selected from each read-write page of data.

The same concept has now been included into the traditional Data step and Procedure step processing.

```
Command window
00001 data new;
00002 set old;
00003 where type='A';
00004 run;
```

It can also be as displayed below.

```
Command window
00001 proc print data=old(where=(type='A'));
00002 run;
```

In the Data step, either as a statement (as shown) or as a data set option, the extraction according to the condition is done well before the data is read into the execution buffer (also known as the Logical Program Data Vector or LPDV).

In the Procedure Step, either as statement or as a data set option (as shown), the data is extracted dynamically during each read of the data set page. The page is the unit of input/output. You can see its size in the Proc Contents output.

The SAS Data Model

The SAS Data Library

The Data Library is the highest unit of organization of SAS files, each file being a member of the library.

The Data Library is a logical level of organization relating to differing data structures within operating systems, but offering the same features within SAS.

For directory-based operating systems such as VMS or DOS, the SAS Data Library relates to the physical directory.

Under CMS, the SAS Data Library is a group of SAS files with the same file type.

Under MVS, the SAS Data Library is a specially formatted MVS data set that may only contain SAS files.

A SAS file is a specially structured file that can be created, organized and maintained by the SAS system.

Note that a file of SAS code is not a SAS file, but is normally held as some external text file.

Each member of a library has a member type.

SAS Data Libraries may contain species of the following types:

- Data
- View
- Catalog
- Access
- Program

Data

This refers to the ordinary SAS data set, where the actual data observations and the descriptor portion of the data set are held in the same physical file.

A physical SAS data set is referred to as a SAS data file.

View

The concept of the SAS data set is extended to cover views. A view may be regarded as a virtual file.

The descriptor portion of the view determines the location and criteria of which data to select.

The actual data may reside in a SAS data set, external database or external file.

A view always dynamically reflects changes in the parent data (which may be a view itself), because only the rules for extraction are stored.

A view can be created with the DBMS software, or with Proc Access or Proc SQL.

Catalog

The SAS catalog can contain many types of entries, for example: graphics; formats; macros; and key definitions.

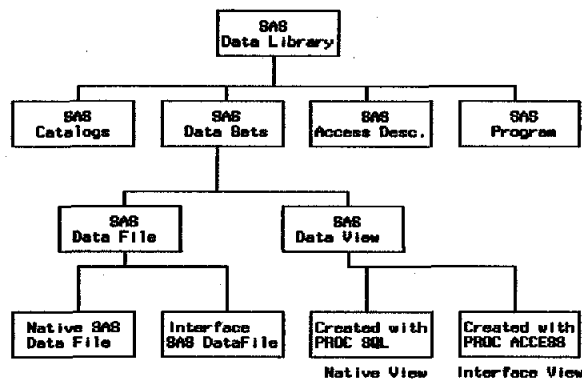
Access

Access descriptors are used to define the data in an external database, which can be used to build a SAS data view.

Proc Access is used to create these.

Program

The following is a compiled form of a SAS data step program.



Interface SAS data files are those where the data is in a file formatted by other software. SAS provides interface engines to read data stored in files formatted by BMDP, SPSS and OSIRIS software.

Following the creation of an Access Module, Proc Access can then be used to create further Views, called Interface Views. They are also known as View Descriptors.

Proc SQL is used to create data files or data views comprised of other data sets and/or views.

Proc SQL is one of the ways to create Indexes on data files.

Capabilities of Proc SQL

Proc SQL primarily can be used to:

1. Extract data stored in:

- SAS data files.
 - Native SAS views created by Proc SQL.
 - Views created from other tables or views held in an external DBMS and accessed with SAS/Access.
- (Note: A single query could take data from any or all of these sources).

In addition, Proc SQL can also do the following:

2. Validate a query-expression without running the code.
3. Create and delete tables, views and indexes.
4. Describe the definition of a native view created with Proc SQL.
5. Add, delete or modify the data in a table.
6. Update the data in the tables and views of an external DMS accessed via SAS/ACCESS views.
7. Change the structure of SAS data files.

The SQL Environment

Proc SQL can be:

- Run interactively, non-interactively or in batch mode.
- Used on both permanent and temporary SAS data sets.
- Combined with any other data/proc step coding as separate steps in a suite of SAS code.
- Used with macro language in the normal way.
- Called from screen control language (SCL).
- Augmented with the usual global SAS statements such as options, titles and footnotes.
- Launched with the SAS query window in SAS/ASSIST.

SQL Statements

An SQL statement is like a verb in an English sentence. It is an action word which tells SQL to do something.

Proc SQL consists of ten statements in addition to the Proc statement. These comprise:

1. **Select** Performs queries
2. **Validate** Validates the syntax of a query
3. **Describe** Shows how a view has been defined
4. **Create** Creates a table (SAS data set), index or view
5. **Drop** Deletes a table, index or view
6. **Update** Changes the values in a table
7. **Insert** Adds rows to a table
8. **Delete** Deletes rows from a table
9. **Alter** Adds, deletes or modifies columns in a table
10. **Reset** Adds, deletes or changes options

The most important is the Select statement.

Also as in an English sentence, the verb (statement) is followed by clauses that explain in more detail just what is expected.

A special keyword is used to begin an SQL clause. These keywords and clauses will be discussed in more detail in the next chapter.

The Select Statement

The Select statement is used to query one or more SAS data sets.

It can be used to:

1. Select columns or variables.
2. Specify an order for the data.
3. Group data.
4. Format variables.
5. Attach labels to variables.

In its simplest form the Select statement consists of two clauses, as displayed below.

```
Command sas>
00001 proc sql;
00002 select clause
00003 from clause;
00004 quit;
```

Example

```
Command sas>
00001 proc sql;
00002 select *
00003 from saved.computer;
00004 quit;
```

CATEGORY	SUPPLIER	TYPE	CPU	DISK	WHOLESALE	RETAIL
1	FLOPPY COMPUTERS	DESKTOP	Pentium	20	580	760
2	FLOPPY COMPUTERS	DESKTOP	Pentium	40	800	800
3	FLOPPY COMPUTERS	DESKTOP	Pentium	100	750	950
4	FLOPPY COMPUTERS	DESKTOP	AMD	40	750	950
5	FLOPPY COMPUTERS	DESKTOP	AMD	100	950	1150
6	FLOPPY COMPUTERS	DESKTOP	AMD	40	950	1150
7	FLOPPY COMPUTERS	DESKTOP	AMD	100	1250	1450
8	FLOPPY COMPUTERS	DESKTOP	Zeon	40	1350	1550
9	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1550	1750
10	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1950	2250
11	FLOPPY COMPUTERS	LAPTOP	Pentium	40	1000	1200
12	FLOPPY COMPUTERS	LAPTOP	AMD	40	1200	1400
13	FLOPPY COMPUTERS	LAPTOP	AMD	100	1400	1600
14	FLOPPY COMPUTERS	LAPTOP	AMD	100	1600	1800
15	FLOPPY COMPUTERS	LAPTOP	AMD	200	1950	2350
16	FLOPPY COMPUTERS	LAPTOP	Pentium	60	1100	1300
17	KETCHUP COMPUTERS	DESKTOP	Pentium	20	1300	1500
18	KETCHUP COMPUTERS	DESKTOP	Pentium	40	1400	1600
19	KETCHUP COMPUTERS	DESKTOP	Pentium	100	1800	1900
20	KETCHUP COMPUTERS	DESKTOP	AMD	20	1200	2000
21	KETCHUP COMPUTERS	DESKTOP	AMD	40	2000	2200
22	KETCHUP COMPUTERS	DESKTOP	AMD	100	2350	2550
23	KETCHUP COMPUTERS	DESKTOP	AMD	20	2450	2650

Points to Note:

1. Clause after Select keyword lists columns. Asterick (*) can be used to print all table columns.
2. Clause after From keyword identifies the table containing data.
3. The Proc SQL statement is loaded into memory and remains resident until another Data or Proc step is run or a Quit; statement is executed.

Subsequent queries or other SQL statements can be run without having to re-submit the Proc SQL statement.

4. Each SQL statement is processed individually. No Run; statement is required.
5. Proc Print style output is produced automatically.
6. There is a line below the variable names and observation numbers are not printed in the output.

This output can be suppressed with the Noprint option on the Proc SQL statement, as displayed below.

```
Command sas>
00001 proc sql noprint;
00002 select;
00003 quit;
00004
```

Usage

Invoke Proc SQL and then use a Select statement.

```
Command ==>
00001 proc sql;
00002
00003
```

```
Command ==>
00001 select *
00002 from saved.computer;
00003
00004
```

```
Command ==>
00001 select *
00002 from saved.house;
00003
00004
```

```
Command ==>
00001 quit;
00002
00003
```

Note that while Proc SQL is running, the Run; statement itself simply produces an interesting note on the Log.

```
Command ==>
00001 run;
00002
00003
```

```
Command ==>
NOTE: PROC SQL statements are executed immediately; The RUN statement has no effect.
00001 run;
00002
00003
```

Examples

The SQL procedure uses the Select statement to perform a wide variety of queries.

There are different clauses in the Select statement:

- The From clause
- The Order By clause
- The Where clause.

Reading Data

The From clause is required. It tells Proc SQL the name(s) of the SAS data set(s) or view(s) containing the data for the query.

Commas separate multiple names. At least one data set/view must be supplied.

Ordering the Result

The Order By clause determines the order in which the rows are displayed in the resulting table. This is comparable to a traditional Sort except that the original data set has not been modified.

```
Command ==>
00001 proc sql;
00002 select *
00003 from saved.computer
00004 order by disk;
00005 quit;
```

To identify a column in the Order By clause the following may be used:

- A column name
- A number corresponding to the position of an item in the select list
- A combination of the above.

Submit the code displayed above.

The output is displayed below.

SALES	SUPPLIER	TYPE	CPU	DISK	WHOLESALE	RETAIL
20	KETCHUP COMPUTERS	DESKTOP	AMD	20	1800	2000
17	KETCHUP COMPUTERS	DESKTOP	Pentium	20	1500	1500
29	KETCHUP COMPUTERS	LAPTOP	Pentium	30	700	850
1	FLOPPY COMPUTERS	DESKTOP	Pentium	20	550	750
25	KETCHUP COMPUTERS	DESKTOP	AMD	20	2400	2650
30	KETCHUP COMPUTERS	LAPTOP	AMD	20	950	1150
12	FLOPPY COMPUTERS	LAPTOP	AMD	40	1200	1400
24	KETCHUP COMPUTERS	DESKTOP	AMD	40	2550	2750
4	FLOPPY COMPUTERS	DESKTOP	AMD	40	750	950
11	FLOPPY COMPUTERS	LAPTOP	Pentium	40	1000	1200
6	FLOPPY COMPUTERS	DESKTOP	AMD	40	950	1150
21	KETCHUP COMPUTERS	DESKTOP	AMD	40	2000	2200
15	KETCHUP COMPUTERS	DESKTOP	Pentium	40	1400	1600
2	FLOPPY COMPUTERS	DESKTOP	Pentium	40	800	900
31	KETCHUP COMPUTERS	LAPTOP	AMD	40	1050	1250
5	FLOPPY COMPUTERS	DESKTOP	Zeon	40	1350	1600
28	KETCHUP COMPUTERS	DESKTOP	Zeon	60	2850	3250
35	KETCHUP COMPUTERS	LAPTOP	AMD	60	1950	2350
18	FLOPPY COMPUTERS	LAPTOP	Pentium	50	1100	1300
3	FLOPPY COMPUTERS	DESKTOP	Pentium	100	750	850
7	FLOPPY COMPUTERS	DESKTOP	AMD	100	1250	1450
14	FLOPPY COMPUTERS	LAPTOP	AMD	100	1800	1900
13	FLOPPY COMPUTERS	LAPTOP	AMD	100	1400	1600
25	KETCHUP COMPUTERS	DESKTOP	AMD	100	2750	2950
22	KETCHUP COMPUTERS	DESKTOP	AMD	100	2350	2550
5	FLOPPY COMPUTERS	DESKTOP	AMD	100	950	1150
10	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1950	2250
19	KETCHUP COMPUTERS	DESKTOP	Pentium	100	1800	1900
9	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1550	1750
27	KETCHUP COMPUTERS	DESKTOP	Zeon	120	3150	3350
34	KETCHUP COMPUTERS	LAPTOP	AMD	120	2300	2450
32	KETCHUP COMPUTERS	LAPTOP	AMD	120	1450	1650
16	FLOPPY COMPUTERS	LAPTOP	AMD	200	1850	2050
26	KETCHUP COMPUTERS	DESKTOP	Zeon	200	3350	3550
36	KETCHUP COMPUTERS	LAPTOP	Zeon	200	3350	3750
35	KETCHUP COMPUTERS	LAPTOP	AMD	200	2850	3350

This code produces the same result as the following.

```
Command ==>
00001 proc sort data = saved.computer out=work.computer;
00002 by disk;
00003 run;
00004 proc print data = work.computer;
00005 run;
```

The advantages of SQL:

- No physical sort has to be performed.
- No physical output file has been produced from this query, simply a listing as output.

Selecting Rows

Selecting rows is done with the Where clause.

```
Command ==>
00001 proc sql;
00002 select *
00003 from saved.computer
00004 where supplier = 'KETCHUP COMPUTERS'
00005 order by disk;
00006 quit;
```

DATA	SUPPLIER	TYPE	CPU	DISK	PRICE	RETAIL
30	KETCHUP COMPUTERS	LAPTOP	AMD	20	950	1150
29	KETCHUP COMPUTERS	LAPTOP	Pentium	20	700	880
20	KETCHUP COMPUTERS	DESKTOP	AMD	20	1800	2000
17	KETCHUP COMPUTERS	DESKTOP	Pentium	20	1300	1500
23	KETCHUP COMPUTERS	DESKTOP	AMD	20	2450	2650
25	KETCHUP COMPUTERS	LAPTOP	AMD	40	1050	1250
21	KETCHUP COMPUTERS	DESKTOP	AMD	40	2000	2200
18	KETCHUP COMPUTERS	DESKTOP	Pentium	40	1400	1600
24	KETCHUP COMPUTERS	DESKTOP	AMD	40	2650	2750
26	KETCHUP COMPUTERS	DESKTOP	Zeon	60	2650	3250
33	KETCHUP COMPUTERS	LAPTOP	AMD	60	1950	2350
19	KETCHUP COMPUTERS	DESKTOP	Pentium	100	1600	1800
22	KETCHUP COMPUTERS	DESKTOP	AMD	100	2350	2550
28	KETCHUP COMPUTERS	DESKTOP	AMD	100	2750	2950
27	KETCHUP COMPUTERS	DESKTOP	Zeon	120	3150	3350
24	KETCHUP COMPUTERS	LAPTOP	AMD	120	2200	2450
12	KETCHUP COMPUTERS	LAPTOP	AMD	120	1450	1650
28	KETCHUP COMPUTERS	DESKTOP	Zeon	200	3550	3750
36	KETCHUP COMPUTERS	LAPTOP	Zeon	200	3550	3750
35	KETCHUP COMPUTERS	LAPTOP	AMD	200	2650	3250

Where Clause Syntax

(These also apply to the Having clause).

Operator	Description
= or EQ	Equals
> or GT	Greater than
< or LT	Less than
>= or GE	Greater or equal to
<= or LE	Less than or equal to
NE or NOT= or ^= or != or <>	Not equal to
IN	Equal to one in a list, e.g. Where disk in (20, 40, 60)
IS	Compares to missing, e.g. Where supplier is missing Where supplier is not null
& or AND	Logical and e.g. Where substr (cpu, 1,3) = '686' And disk = 20
^ or ~ or NOT	Logical not
OR or or !	Logical or
LIKE '%' or '_'	Compares two patterns: Where type like 'desk%' Desk followed by anything Where cpu like '386_x' Replaces any one character
? or CONTAINS	Compares two patterns: Where type? 'top' Top in any part of word Desktop, laptop, tophat
BETWEEN	Selects inclusive ranges: Where disk between 40 and 100
= *	'Sounds like' operator
FUNCTIONS	SAS functions are permitted

Notes

- Computed columns cannot be referred to by name in a Where clause. They must be recalculated for the Where clause.
- = is normally the symbol used for Not Equals on an IBM mainframe.
- ^= is normally the symbol used for Not Equals on ASCII based machines.
- NOT= and NE and <> can also be used for Not Equals.
- The 'Sounds Like' operator uses the Soundex algorithm. This is usually used to search for variants in names, for example telephone directory applications, but it is not perfect.
- The Between operator includes values defined in the range.
- Is Null and Is Missing are equivalent and match with all missing values including special missing values.
- SAS functions are supported except for Lag, Dif and Sound. These supported SAS functions should not be confused with SQL Summary functions seen in a later section of this chapter.

Controls and Enhancements

The Feedback Option

The Feedback option can be used to list all columns represented by an asterisk in a Select statement.

Submit the code below.

```

Command ==>
00001 ***The Feedback Option;
00002
00003 proc sql feedback;
00004 select *
00005 from saved.computer;
00006 quit;

```

Columns are listed in the log.

```

00001 ***The Feedback Option;
00002
00003 proc sql feedback;
00004 select *
00005 from saved.computer;
00006 quit;

NOTE: Statement transformer id:
select COMPUTER.CATHUM, COMPUTER.SUPPLIER, COMPUTER.TYPE, COMPUTER.CPU, COMPUTER.DISK,
COMPUTER.UNCLE2M, COMPUTER.RETAIL
from SAVED.COMPUTER;

NOTE: Writing HTML Body file: sashtml.htm
00007 quit;
NOTE: PROCEDURE SQL used:
real time      0.31 seconds
cpu time       1.06 seconds

```

The HTML output follows.

CRTNUM	SUPPLIER	TYPE	CPU	DISK	WHOLESALE	RETAIL
1	FLOPPY COMPUTERS	DESKTOP	Pentium	20	550	750
2	FLOPPY COMPUTERS	DESKTOP	Pentium	40	600	800
3	FLOPPY COMPUTERS	DESKTOP	Pentium	100	750	950
4	FLOPPY COMPUTERS	DESKTOP	AMD	40	750	950
5	FLOPPY COMPUTERS	DESKTOP	AMD	100	850	1150
6	FLOPPY COMPUTERS	DESKTOP	AMD	40	950	1150
7	FLOPPY COMPUTERS	DESKTOP	AMD	100	1250	1450
8	FLOPPY COMPUTERS	DESKTOP	Zeon	40	1350	1650
9	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1550	1750
10	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1650	2250
11	FLOPPY COMPUTERS	LAPTOP	Pentium	40	1000	1200
12	FLOPPY COMPUTERS	LAPTOP	AMD	40	1200	1400
13	FLOPPY COMPUTERS	LAPTOP	AMD	100	1400	1600
14	FLOPPY COMPUTERS	LAPTOP	AMD	100	1600	1900
15	FLOPPY COMPUTERS	LAPTOP	AMD	200	1850	2350
16	FLOPPY COMPUTERS	LAPTOP	Pentium	60	1100	1300
17	KETCHUP COMPUTERS	DESKTOP	Pentium	20	1500	1800

The list output will appear as follows.

CRTNUM	SUPPLIER	TYPE	CPU	DISK	WHOLESALE	RETAIL
1	FLOPPY COMPUTERS	DESKTOP	Pentium	20	550	750
2	FLOPPY COMPUTERS	DESKTOP	Pentium	40	600	800
3	FLOPPY COMPUTERS	DESKTOP	Pentium	100	750	950
4	FLOPPY COMPUTERS	DESKTOP	AMD	40	750	950
5	FLOPPY COMPUTERS	DESKTOP	AMD	100	850	1150
6	FLOPPY COMPUTERS	DESKTOP	AMD	40	950	1150
7	FLOPPY COMPUTERS	DESKTOP	AMD	100	1250	1450
8	FLOPPY COMPUTERS	DESKTOP	Zeon	40	1350	1650
9	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1550	1750
10	FLOPPY COMPUTERS	DESKTOP	Zeon	100	1650	2250
11	FLOPPY COMPUTERS	LAPTOP	Pentium	40	1000	1200
12	FLOPPY COMPUTERS	LAPTOP	AMD	40	1200	1400
13	FLOPPY COMPUTERS	LAPTOP	AMD	100	1400	1600
14	FLOPPY COMPUTERS	LAPTOP	AMD	100	1600	1900
15	FLOPPY COMPUTERS	LAPTOP	AMD	200	1850	2350
16	FLOPPY COMPUTERS	LAPTOP	Pentium	60	1100	1300
17	KETCHUP COMPUTERS	DESKTOP	Pentium	20	1500	1800
18	KETCHUP COMPUTERS	DESKTOP	Pentium	40	1400	1600
19	KETCHUP COMPUTERS	DESKTOP	Pentium	100	1600	1800
20	KETCHUP COMPUTERS	DESKTOP	AMD	20	1800	2000
21	KETCHUP COMPUTERS	DESKTOP	AMD	40	2000	2200
22	KETCHUP COMPUTERS	DESKTOP	AMD	100	2350	2550
23	KETCHUP COMPUTERS	DESKTOP	AMD	20	2450	2650
24	KETCHUP COMPUTERS	DESKTOP	AMD	40	2550	2750
25	KETCHUP COMPUTERS	DESKTOP	AMD	100	2750	2950
26	KETCHUP COMPUTERS	DESKTOP	Zeon	60	2850	3050
27	KETCHUP COMPUTERS	DESKTOP	Zeon	120	3150	3350
28	KETCHUP COMPUTERS	DESKTOP	Zeon	200	3350	3550
29	KETCHUP COMPUTERS	LAPTOP	Pentium	20	700	850
30	KETCHUP COMPUTERS	LAPTOP	AMD	20	950	1150
31	KETCHUP COMPUTERS	LAPTOP	AMD	40	1050	1250
32	KETCHUP COMPUTERS	LAPTOP	AMD	120	1450	1850
33	KETCHUP COMPUTERS	LAPTOP	AMD	60	1850	2350
34	KETCHUP COMPUTERS	LAPTOP	AMD	120	2250	2450
35	KETCHUP COMPUTERS	LAPTOP	AMD	200	2550	2850

Selecting Columns

Using the column names from the SAS file allows selection of the columns in the report.

```

Command ==>
00001 proc sql;
00002   select type, disk, retail
00003   from saved.computer
00004   where supplier = 'KETCHUP COMPUTERS'
00005   order by disk;
00006 quit;

```

TYPE	DISK	RETAIL
LAPTOP	20	1150
LAPTOP	20	850
DESKTOP	20	2000
DESKTOP	20	1500
DESKTOP	20	2850
LAPTOP	40	1250
DESKTOP	40	3300
DESKTOP	40	1900
DESKTOP	40	2750
DESKTOP	60	3250
LAPTOP	60	3350
DESKTOP	100	1800
DESKTOP	100	2550
DESKTOP	100	2350
DESKTOP	120	3350
LAPTOP	120	2450
LAPTOP	120	1950
DESKTOP	200	3550
LAPTOP	200	3750
LAPTOP	200	3350

This is the equivalent of the following.

```

Command ==>
00001 proc sort data = saved.computer out=work.computer;
00002   by disk;
00003 run;
00004 proc print data=work.computer
00005   (where=(supplier='KETCHUP COMPUTERS'));
00006   var type disk retail;
00007 run;

```

The advantage of SQL is that there is no need to sort the data.

Note that commas separate the list of column names.

select type, disk, retail

All lists of column names and table names require commas in SQL syntax.

Calculating Columns

Columns can be calculated by assigning an expression to an item name.

```

Command ==>
00001 proc sql;
00002   select type,
00003         disk,
00004         retail,
00005         retail*.06 as Tax
00006   from saved.computer
00007   where supplier = 'KETCHUP COMPUTERS'
00008   order by disk;
00009 quit;
00010

```

Notice the structure:

expression As variable

Opposed to the traditional SAS

variable = expression;

TYPE	DISK	RETAIL	TAX
LAPTOP	20	1150	69
LAPTOP	20	850	51
DESKTOP	20	2000	120
DESKTOP	20	1500	90
DESKTOP	20	2850	169
LAPTOP	40	1250	75
DESKTOP	40	3300	132
DESKTOP	40	1900	96
DESKTOP	40	2750	165
DESKTOP	60	3250	195
LAPTOP	60	3350	201
DESKTOP	100	1800	108
DESKTOP	100	2550	153
DESKTOP	100	2350	141
DESKTOP	120	3350	201
LAPTOP	120	2450	147
LAPTOP	120	1950	99
DESKTOP	200	3550	213
LAPTOP	200	3750	225
LAPTOP	200	3350	201

Notice that the traditional equivalent would now require an additional data step.

```

Command ==>
00001 data new;
00002   set saved.computer;
00003   Tax = retail *.06;
00004 run;
00005 proc sort data = new;
00006   by disk;
00007 run;
00008 proc print data = new (where=(supplier='KETCHUP COMPUTERS'));
00009   var type disk retail Tax;
00010 run;
00011

```

Formatting Values - A Column Modifier

The format option appears after the column name, not as a separate statement.

```
Command ==>
00001 proc sql;
00002   title 'Ketchup Computers, TAX element of prices';
00003   select type,
00004          disk,
00005          retail,
00006          retail*.06 as tax format=dollar7.2
00007   from saved.computer
00008  where supplier='KETCHUP COMPUTERS'
00009  order by disk;
00010 quit;
00011
```

Ketchup Computers, TAX element of prices

TYPE	DISK	RETAIL	TAX
LAPTOP	20	1150	\$69.00
LAPTOP	20	850	\$51.00
DESKTOP	20	2000	\$120.00
DESKTOP	20	1000	\$60.00
DESKTOP	20	2850	\$171.00
LAPTOP	40	1250	\$75.00
DESKTOP	40	2200	\$132.00
DESKTOP	40	1600	\$96.00
DESKTOP	40	2750	\$165.00
DESKTOP	50	3250	\$195.00
LAPTOP	60	2450	\$147.00
DESKTOP	100	1800	\$108.00
DESKTOP	100	2500	\$150.00
DESKTOP	100	2950	\$177.00
DESKTOP	120	3300	\$201.00
LAPTOP	120	2450	\$147.00
LAPTOP	120	1850	\$99.00
DESKTOP	200	3550	\$213.00
LAPTOP	200	3750	\$225.00
LAPTOP	200	3500	\$210.00

This is a SAS enhancement to standard SQL.

Labeling Columns - A Column Modifier

Column headings can be changed with the Label option.

```
Command ==>
00001 proc sql;
00002   title 'Ketchup Computers, TAX element of prices';
00003   select type label='Computer Type',
00004          retail label='Retail Price' format=comma8.2,
00005          retail*.06 as Tax
00006   from saved.computer
00007  where supplier='KETCHUP COMPUTERS'
00008  order by disk;
00009 quit;
00010
```

Ketchup Computers, TAX element of prices

Computer Type	Retail Price	Tax
LAPTOP	1150.00	84
LAPTOP	850.00	51
DESKTOP	2000.00	120
DESKTOP	1000.00	60
DESKTOP	2850.00	159
LAPTOP	1250.00	75
DESKTOP	2200.00	132
DESKTOP	1600.00	96
DESKTOP	2750.00	165
DESKTOP	3250.00	195
LAPTOP	2450.00	147
DESKTOP	1800.00	108
DESKTOP	2500.00	150
DESKTOP	2950.00	177
DESKTOP	3350.00	201
LAPTOP	2450.00	147
LAPTOP	1850.00	99
DESKTOP	3550.00	213
LAPTOP	3750.00	225
LAPTOP	3500.00	201

This is a SAS enhancement to standard SQL.

Using SAS Data Step Functions

Data Step functions can be used in these expressions as noted earlier in this chapter.

```
Command ==>
00001 options nodate nonumber;
00002 title 'Charges raised for car hire';
00003 proc format;
00004   value $model 'F' = 'Fiesta'
00005               'C' = 'Escort'
00006               'D' = 'Orion'
00007               'S' = 'Sierra'
00008               'G' = 'Grenade';
00009 run;
00010 proc sql;
00011   select substr(carkey,2,1) format=$model.,
00012          cuetkey label='Customer Code',
00013          daychg label='Daily Charge',
00014          (enddate - stdate + 1) as timekept
00015          label='Days Hired',
00016          (enddate - stdate + 1)*daychg,
00017          round(((enddate - stdate + 1)*daychg*.06),0.01)
00018          as Tax
00019   from saved.carhire
00020  order by S desc;
00021 quit;
```

Notes:

All SAS Data Step functions are supported except Lag, Dif and Sound.

Functions can be used to derive calculated columns as shown by the Substr function above.

They can also be used to change calculated values, as shown by the Round function.

Derived columns need not be given aliases using the As syntax.

Aliases cannot be used in further calculations.

For example, review the expression in the round function above.

$(enddate - stdate + 1) * daychg$

If we were to replace it with the following:

$timekept * daychg$

There would be an error since timekept would not be found.

The Order By clause can use the ordinal position of the column.

In the example above, the fifth column is used. This is how ordering is done on a calculated column with no alias.

Here the order is to be descending, coded Desc.

Charges raised for car hire

Customer Code	Daily Charge	Days Hired	Tax
Escort MA	50	30	900
Escort AM	75	10	750
Fiesta MA	40	8	640
Escort OFF	40	10	400
Fiesta MA	25	12	300
Fiesta MA	25	12	300
Fiesta TT	35	8	280
Sierra TCL	35	2	110
Escort NL	25	4	100
Escort NL	25	3	75
Escort NL	25	2	50
Orion FS	5	3	15

Calculated Syntax

In addition to using SQL to calculate columns, sometimes recalculations are required. A simple situation might require a totaling of the retail price and tax. Logically, the code would look like the following.

```
Command ===>
00001 * Syntax Error produced for example purposes.
00002 proc sql;
00003   select catnum,
00004          retail,
00005          retail*.05 as tax,
00006          retail + tax as total
00007   from saved.computer;
00008 quit;
```

Note that this code produces an error.

```
Command ===>
00001 proc sql;
00002   select catnum,
00003          retail,
00004          retail*.05 as tax,
00005          retail + tax as total
00006   from saved.computer;
ERROR: The following columns were not found in the contributing tables: tax.
00007 quit;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE SQL used:
      real time      0.43 seconds
      cpu time       0.02 seconds
```

In order to produce the correct output, repeat the entire calculation of tax in the total or use the Calculated syntax.

```
Command ===>
00001 proc sql;
00002   select catnum,
00003          retail,
00004          retail*.05 as tax,
00005          retail + calculated tax as total
00006   from saved.computer;
00007 quit;
```

CATNUM	RETAIL	TAX	TOTAL
1	750	45	795
2	600	48	648
3	950	57	1007
4	950	57	1007
5	1150	69	1219
6	1150	69	1219
7	1450	87	1537
8	1550	93	1643
9	1750	105	1855
10	2250	135	2385
11	1200	72	1272
12	1400	84	1484
13	1600	96	1696
14	1800	114	1914
15	2350	141	2491
16	1500	75	1575

The word Calculated is required prior to reusing a column that was just created in the select statement.

Distinct Values

Distinct is used to eliminate duplicate rows.

```
Command ===>
00001 proc sql;
00002   title 'Which disk types are sold by Ketchup?';
00003   select distinct disk
00004   from saved.computer
00005   where suppliers='KETCHUP COMPUTERS';
00006 quit;
```

Which disk types are sold by Ketchup?

20
40
60
100
120
200

Traditional SAS programming would involve the following.

```
Command ===>
00001 title 'Which disk types are sold by Ketchup?';
00002 proc sort data=saved.computer(wheras=(suppliers='KETCHUP COMPUTERS'));
00003   out=work.unique(keep=disk) nodupkey;
00004   by disk;
00005 run;
00006 proc print data=work.unique;
00007 run;
```

Selecting Data with Distinct

The Distinct keyword can be used to select data for all combinations of columns.

```
Command ===>
00001 proc sql;
00002   title 'All combinations of disk and type';
00003   select distinct disk label='Hard Disk Size',
00004          type label='Computer Type'
00005   from saved.computer
00006   where suppliers='KETCHUP COMPUTERS'
00007   order by disk;
00008 quit;
```

All combinations of disk and type

20 DESKTOP
20 LAPTOP
40 DESKTOP
40 LAPTOP
60 DESKTOP
60 LAPTOP
100 DESKTOP
120 DESKTOP
120 LAPTOP
200 DESKTOP
200 LAPTOP

The Distinct keyword applies to all the columns listed. Each unique combination of column values is returned.

Notice that a Title statement was used to enhance the output.

Title and Footnote statements must precede the Select statement.

Data/Proc Step methods would use two By variables.

Note the effect on the inner variable (type) when the outer variable (disk) changes value.

```
Command ===>
00001 title 'All combinations of disk and type';
00002 proc sort data=saved.computer(wheras=(suppliers='KETCHUP COMPUTERS');
00003   out=work.unique(keep=(disk type) nodupkey;
00004   by disk type;
00005 run;
00006 proc print data=work.unique;
00007 run;
```

Syntax Check Using the Validate Statement

The Validate statement is used to test a Select statement.

It checks that the listed column names are correct.

It also tests the query syntax without actually executing the query.

Error messages are printed in the Log.

```
Command ==> |
00001 proc sql;
00002 validate
00003 select distinct disk label='Hard Disk Size',
00004 type label='Computer Type'
00005 from saved.computer
00006 where supplier='KETCHUP COMPUTERS'
00007 order by disk;
00008 quit;
```

```
Command ==> |
NOTE: PROC SQL statement has valid syntax.
```

Syntax Errors

This syntax error takes place because the Order By option is not the last clause on the Select statement.

```
Command ==> |
00001 proc sql;
00002 validate
00003 select distinct disk label = 'Hard Disk Size',
00004 type label = 'Computer Type'
00005 from saved.computer
00006 order by disk
00007 where supplier = 'KETCHUP COMPUTERS';
00008 quit;
```

```
Command ==> |
238 validate
239 select distinct disk label = 'Hard Disk Size',
240 type label = 'Computer Type'
241 from saved.computer
242 order by disk
243 where supplier = 'KETCHUP COMPUTERS';
22 ERROR 22-322: Syntax error, expecting one of the following: ;, |, ||, &, (, =, >, <, AND, ASC, ASCENDING, BETWEEN, CONTAINS, DESC, DESCENDING, NOTIN, OR, ^, ^=, ||, ||, ^, ^=.
ERROR 202-322: The option or parameter is not recognized and will be ignored.
```

Make sure that the Order By statement is the last option on the Select statement.

Syntax Check Using the Noexec Option

The Noexec option can be used to check for correct syntax in several statements including Select, Create, Delete, Alter, Drop, Insert, Reset and Update.

The SQL code is not actually executed. Error messages are printed in the Log.

```
Command ==> |
00001 proc sql noexec;
00002 quit;
```

Analysis on Groups

Summary Functions

A series of Summary functions are provided to analyze columns. A complete list of these functions is provided later in the chapter.

Single Column in a Summary Function

If a single column is specified in a Summary function, the statistic is calculated down the column just like a traditional SAS Proc would operate.

```
Command ==> |
00001 proc sql;
00002 select mean(retail) as suprice
00003 from saved.computer;
00004 quit;
```

All combinations of disk and type

1929.167

This is the equivalent of the following.

```
Command ==> |
00001 proc means data = saved.computer mean;
00002 var retail;
00003 run;
```

Multiple Columns in a Summary Function

If more than one column is specified in a Summary function, the calculation is done across the columns for each row.

```
Command ==> |
00001 proc sql;
00002 select retail format= comma10.2,
00003 retail * .08 as TAX format = comma10.2,
00004 sum(retail,retail*.08) as GROSS format = comma10.2
00005 from saved.computer;
00006 quit;
```

750.00	60.00	780.00
800.00	64.00	848.00
950.00	76.00	1007.00
950.00	76.00	1007.00
1150.00	92.00	1238.00
1150.00	92.00	1238.00
1450.00	116.00	1587.00
1450.00	116.00	1587.00
1550.00	124.00	1696.00
1750.00	140.00	1896.00
2250.00	180.00	2395.00
2200.00	176.00	2272.00
1700.00	136.00	1864.00
1500.00	120.00	1680.00
1400.00	112.00	1584.00
2350.00	188.00	2491.00
1300.00	104.00	1476.00

Single Summary Function with Other Individual Columns – Remerging

If a single Summary function is part of a Select statement where other individual columns have been selected, the resulting Summary statistic is combined with all the individual table rows.

This is known as Remerging.

```
Command ==> |
00001 proc sql;
00002 select cpu,
00003 disk,
00004 (retail-wholesale)
00005 as profit label='Profit',
00006 mean(retail-wholesale)
00007 as auprofit label = 'Average Profit',
00008 (retail-wholesale) - mean(retail-wholesale)
00009 as diff label = 'Difference'
00010 from saved.computer
00011 where supplier contains 'FLOPPY';
00012 quit;
```

```
Command ==> |
275 select cpu,
276 disk,
277 (retail-wholesale)
278 as profit label='Profit',
279 mean(retail-wholesale)
280 as auprofit label = 'Average Profit',
281 (retail-wholesale) - mean(retail-wholesale)
282 as diff label = 'Difference'
283 from saved.computer
284 where supplier contains 'FLOPPY';
NOTE: The query requires remerging summary statistics back with the original data.
285 quit;
```

CPU	DISK	Price	Average Price	Difference
Pentium	20	200	231.25	-31.25
Pentium	40	200	251.25	-51.25
Pentium	100	200	281.25	-81.25
AMD	40	200	251.25	-51.25
AMD	100	200	281.25	-81.25
AMD	40	200	231.25	-31.25
AMD	100	200	251.25	-51.25
Zeon	40	200	231.25	-31.25
Zeon	100	200	251.25	-51.25
Zeon	100	300	281.25	-81.25
Pentium	40	200	231.25	-31.25
AMD	40	200	251.25	-51.25
AMD	100	200	281.25	-81.25
AMD	100	300	251.25	-81.25
AMD	200	500	281.25	-281.25
Pentium	60	200	231.25	-31.25

Data Step Equivalent

This also can be accomplished with a Data/Proc step, as illustrated here.

Proc Means/Summary can be used to create a one-observation, one-variable data set.

This is then read into the data step alongside Saved.Computer.

Alternatively, it can be read in two passes of the data in the same data step.

```

Command ===>
00001 data work.new;
00002 retain supprofit 0;
00003 if _n_ = 1 then do;
00004   go until(end);
00005   set saved.computer end=end;
00006   if substr(supplier,1,5) = 'FLOPPY';
00007   count = 1;
00008   accum = retail;
00009   accum = wholesal;
00010   supprofit = (accum-accumn)/count;
00011 end;
00012 and;
00013 set saved.computer(
00014   where=(substr(supplier,1,5) = 'FLOPPY'));
00015 profit = retail - wholesal;
00016 diff = profit - supprofit;
00017 run;
00018 proc print data=work.new;
00019 var cpu disk profit supprofit diff;
00020 label profit = 'Profit';
00021 supprofit = 'Average Profit';
00022 diff = 'Difference';
00023 run;

```

Count Function

Count(*) gives the number of table rows (data set observations).

Count is the only function that allows an asterisk (*) as an argument.

The following examples use a SAS data set with details of computers at a computer retail store. These examples answer various business questions about computer sales.

Example 1: How many total rows in a table? What is the average retail price?

```

Command ===>
00001 proc sql;
00002 select count(*) as no_rows
00003 from saved.computer;
00004 select sum(retail)/count(*) as average
00005 from saved.computer;
00006 quit;

```

no_rows	36
average	1929.167

When used with other arguments or modifiers the Count function can be used to calculate the number of rows for several situations.

Example 2: How many Ketchup Computers in table?

```

Command ===>
00001 proc sql;
00002 select count(*)
00003 from saved.computer
00004 where supplier='KETCHUP COMPUTERS';
00005 quit;

```

The output is displayed, as follows.

	20
--	----

Example 3: How many types of suppliers?

```

Command ===>
00001 proc sql;
00002 select distinct supplier
00003 from saved.computer;
00004 quit;

```

The output is displayed below.

SUPPLIER
FLOPPY COMPUTERS
KETCHUP COMPUTERS

Example 4: How many different suppliers?

```

Command ===>
00001 proc sql;
00002 select count(distinct supplier)
00003 from saved.computer;
00004 quit;

```

The output is displayed below.

	2
--	---

Group By and Having Clauses

Analyzing groups of data is performed using the Group By and Having clauses with the Select statement.

A Group By clause is used to analyze data based on group membership. With a Group By clause the analysis is performed for each group. Without a Group By clause the analysis is performed on the entire table.

Every column you list in a Group By clause also has to be part of the Select list.

A Having clause is used to operate on groups of data. In a Having clause groups are first formed and then some operation is performed based on those groupings.

Several types of analyses can be performed using the Group By and Having clauses.

These include the following.

Style	Select Statement	Result
1. Distinct summary on a group	Group By variable and a summary function on a second variable	Equivalent of By statement. Has one row each value of Group By variable. Data calculated for each Group By value.
2. Group & table summaries used in query of all records	Group By variable and a summary function on one or more variables	Has one row for each original row, subject to Where or Having clauses. Data calculated for each Group By value.
3. Group results only	Group By variable and several variables all have summary function	Has one row for each value of Group By variable. Data calculated for each Group By value
4. Group summary used in query	Group By variable and several variables, with the summary function on Having not Select	Has one row for each row in the original file, subject to the Having clause. Data is calculated for each Group By value.
5. No summary	Group By variable and several variables, no summary function on Select or Having	Group By translated into an Order By option. Has one row for each value in the original table, subject to Where and Having clauses.

Style 1: Sub-Group Analysis – Group By Clause

Example 1: Compare Proc Means analysis using Class statement with Proc SQL.

```

Command ===
00001 proc means data=saved.computer mean nway;
00002   class disk;
00003   var retail;
00004 run;

```

The output is displayed below.

The MEANS Procedure

ANALYSIS VARIABLE RETAIL

DISK	N	Obs	Mean
20	6		1483.33
40	10		1485.00
60	3		2300.00
100	10		1835.00
120	3		2483.33
200	4		3250.00

```

Command ===
00001 proc sql;
00002   select disk,
00003     mean(retail) as avgret
00004   from saved.computer
00005   group by disk;
00006 quit;

```

DISK	MEAN
20	1483.333
40	1485
60	2300
100	1835
120	2483.333
200	3250

Simple summarization by a class variable (e.g. disk) creates summaries (e.g. mean) of an analysis variable (e.g. retail).

There is one row for each value of the Group By variable.

Example 2: Compare data step analysis with Proc SQL

With traditional SAS programming the data step can be used to calculate summary statistics for variables as a function of group membership as follows.

```

Command ===
00001 title 'What is the total retail price for each disk sold?';
00002 proc sort data=saved.computer out=sorted;
00003   by disk;
00004 run;
00005 data unique(keep=disk totret);
00006   set sorted(keep=disk supplier retail);
00007   by disk;
00008   if first.disk then totret=0;
00009   totret+retail;
00010   if last.disk;
00011   where supplier='KETCHUP COMPUTERS';
00012 run;
00013 proc print data=unique;
00014 run;

```

With SQL the summary function - Sum() and Group By clause are used to perform the same analysis:

```

Command ===
00001 proc sql;
00002   title 'Total retail price for each disk type sold';
00003   select disk,
00004     sum(retail) as totret
00005   from saved.computer
00006   where supplier='KETCHUP COMPUTERS'
00007   group by disk;
00008 quit;

```

The output is displayed below.

Total retail price for each disk type sold		
disk	price	total
30	8150	
30	7800	
50	5800	
100	7300	
120	7450	
200	10950	

Style 2: Compare Individual Values to Group Averages

The following compares individual values to group averages using Group By and Having Clauses together.

Example 1: Traditional SAS Programming.

Often individual values need to be compared with the average value for the group, instead of the whole data set.

The following traditional SAS programming performs this analysis.

```

Command ===>
00001 proc sort data=saved.demograp out=demograp;
00002   by gender;
00003 run;
00004 proc means data=demograp mean noprint;
00005   by gender;
00006   var salary;
00007   output out=state mean=avgsl;
00008 run;
00009 data lowest highest;
00010   set demograp;
00011   by gender;
00012   if first.gender then set state;
00013   if salary < avgsl then output lowest;
00014   else output highest;
00015 run;
00016 proc print data=lowest;
00017   title 'Employees with lower than average salaries';
00018 run;
00019 proc print data=highest;
00020   title 'Employees with higher than average salaries';
00021 run;

```

Example 2: Using the Having and Group By clause.

The following Proc SQL code makes use of the Having clause with a Group By clause to perform the same analysis.

```

Command ===>
00001 proc sql;
00002   select gender, status, salary, avg(salary) as avgsl
00003   from saved.demograp
00004   group by gender
00005   having salary > avg(salary);
00006 quit;

```

The Having clause operates on an entire group of data rather than on individual rows.

A Group By clause usually precedes a Having clause defining the group the Having clause evaluates.

In this example, the Group By clause specifies that the data be grouped based on the values of the variable Gender. As a result, two groups will be formed.

The Having clause then operates on these groups. In this case, the Having clause compares the individual value of salary for each row to the average salary for the appropriate group.

```

NOTE: The query requires rearranging summary statistics back with the original data.
NOTE: Writing HTML Body file: sashtml.htm

```

The output is displayed below.

gender	status	salary	avgsl
F	M	30000	10980.95
F	M	19000	10980.95
F	M	18000	10980.95
F	M	18000	10980.95
F	M	15000	10980.95
F	M	15000	10980.95
F	S	30000	10980.95
F	M	15000	10980.95
F	SEP	18000	10980.95
F	M	30000	10980.95
F	S	35000	10980.95
M	M	20000	12007.14
M	M	20000	12007.14
M	M	15000	12007.14
M	M	40000	12007.14

Avgsl represents the different averages for the two gender groups.

Style 3: Sub-Group Analysis for a Subset of Observations

This is accomplished by incorporating the Group By and Where Clauses together.

Consider the most recent output.

gender	status	salary	avgsl
F	M	30000	10980.95
F	M	19000	10980.95
F	M	18000	10980.95
F	M	18000	10980.95
F	M	15000	10980.95
F	M	15000	10980.95
F	S	30000	10980.95
F	M	15000	10980.95
F	SEP	18000	10980.95
F	M	30000	10980.95
F	S	35000	10980.95
M	M	20000	12007.14
M	M	20000	12007.14
M	M	15000	12007.14
M	M	40000	12007.14

This displays the individual rows that have a value for salary that is greater than the average salary for the appropriate Male or Female group.

Change the code in order to calculate the average salary and average number of cars for Females and Males, but only for individuals with a salary greater than 10,000.

The following code performs this analysis.

```

Command ===>
00001 proc sql;
00002   select avg(salary) label='Average Salary'
00003   format=B.2,
00004   avg(cars) label='Average Number of Cars'
00005   format=3.1,
00006   gender
00007   from saved.demograp
00008   where salary > 10000
00009   group by gender;
00010 quit;

```

Average Number of Days Sick		
Age Group	Gender	Average
1970000	14 F	
2086333	13 M	

The Where clause is used to select observations before the averages are calculated.

The Group By clause determines the groupings for the analysis.

Style 4: Compare Individual Values to Group Averages and Summarize the Result

This is accomplished by using the Group By and Having Clauses Together.

In this example a summary function does not appear on the Select list but the Having clause does have a summary function.

The Group By clause is again used to form analysis groups.

Each individual's salary is compared to the average salary for the appropriate gender group.

```
Command ===>
00001 proc sql;
00002 select gender, status, salary
00003 from sash.data.demograp
00004 group by gender
00005 having salary > avg(salary);
00006 quit;
```

gender	status	salary
F	W	30000
F	M	15000
F	M	16000
F	M	18000
F	M	15000
F	S	30000
F	M	15000
F	SEP	18000
F	M	30000
F	S	13000
M	M	23000
M	M	23000
M	M	12300
M	M	40000

This analysis is identical to that performed for Style 2.

The difference is that in this example the Summary function is not in the Select list and therefore is not displayed.

Style 5: No Summary Function

In this example grouping does not occur because a Summary function does not appear in the Select list nor in the Having clause.

Proc SQL translates the Group By clause into an Order By clause and an error message is printed in the Log.

```
Command ===>
00001 proc sql;
00002 select gender, status, salary
00003 from sash.data.demograp
00004 group by status
00005 having salary > 10000;
00006 quit;
```

```
Command ===>
WARNING: A GROUP BY clause has been transformed into an ORDER BY clause because neither the
HAVING clause of the associated table-expression referenced a summary function.
```

The Log indicates that the Group By clause has been converted to an Order By clause.

What the Log does not indicate is that the Having clause is now treated like a Where clause.

gender	status	salary
M	M	23000
F	M	19000
F	M	30000
M	M	12300
F	M	19000
F	M	15000
M	M	12000
M	M	40000
M	M	23000
F	M	16000
F	M	13000
F	S	13000
F	S	30000
F	SEP	16000
M	SEP	12000
F	W	30000

Sub-Queries/Inner Queries

The result of a query may be embedded inside further queries.

These embedded queries are known as Sub-queries or Inner queries.

A Sub-query has the following attributes:

1. It consists of a Select statement that is written beneath another Select statement.
2. It is enclosed in parentheses.
3. It executes before the outer query (first Select statement that appears in query).
4. It typically provides results that are used as part of a Where or Having clause.
5. It produces either a single value or a set of values, which are then used by the outer query.

For illustration purposes examine the average profit for each CPU type.

```
Command ===>
00001 proc sql;
00002 select cpu,
00003 avg(retail - wholesale) as profit
00004 from sash.data.computer
00005 group by cpu;
00006 quit;
```

cpu	profit
AMD	142.5
Pentium	194.4444
Zen	251.1429

Analysis indicates that some of the CPU types are more profitable than other ones.

Example 1: Sub-query that returns a single value.

By what method can the average profit for each CPU type be compared to the average profit for the entire data set?

Do this by using the Having option and a Sub-query.

```
Command ===>
00001 proc sql;
00002   select cpu,
00003          avg(retail-wholesale) as profit /* < OUTER QUERY */
00004   from saved.computer
00005  group by cpu
00006  having profit >
00007         (select avg(retail-wholesale) /* < SUBQUERY (or INNER QUERY)
00008          from saved.computer);
00009 quit;
```

In this example, only the CPU types with profit margins greater than the overall average are displayed.

CPU	profit
AMD	242.5
Zeon	257.1429

The Inner query code consists of the following.

```
Command ===>
00001 title Result of INNER QUERY;
00002 proc sql;
00003   select avg(retail-wholesale)
00004   from saved.computer;
00005 quit;
```

The result of this Inner query is a single value, the average profit for the entire data set.

Result of INNER QUERY	
	233.3333

The Inner query result is then used by the Outer query to produce the desired analysis.

One way to perform this analysis using traditional data/proc steps is as follows.

```
Command ===>
00001 data new;
00002   set saved.computer;
00003   profit = retail - wholesale;
00004 run;
00005 proc summary data=new;
00006   class cpu;
00007   var profit;
00008   output out=new2 mean=meanprofit;
00009 run;
00010 data new3;
00011   retain totavprf;
00012   set new2;
00013   if _type_ = 0 then totavprf = meanprofit;
00014   if meanprofit > totavprf;
00015 run;
00016 proc print data=new3;
00017 run;
```

Obs	totavprf	CPU	_TYPE_	FREQ	meanprofit
1	233.333	AMD	1	20	242.500
2	233.333	Zeon	1	7	257.143

This demonstrates the power of the SQL query where one query replaces three data/proc steps.

Example 2: Sub-query that returns several values.

The next example shows an Inner query that results in a list of values, which are then used by the Outer query as part of a Where clause with an In operator.

The Inner query code consists of the following code.

```
Command ===>
00001 proc sql;
00002   select distinct cpu
00003   from saved.computer
00004  where retail < 1200;
00005 quit;
```

This Inner query returns two values, those CPU types that have a retail value less than 1200.

Result of INNER QUERY	
	AMD
	Pentium

The Full Query code consists of the following.

```
Command ===>
00001 proc sql;
00002   select cpu,
00003          disk,
00004          retail - wholesale as profit
00005   from saved.computer
00006  where cpu in
00007         (select distinct cpu
00008          from saved.computer
00009          where retail < 1200);
00010 quit;
```

The full query result is displayed as follows.

CPU	disk	profit
Pentium	20	200
Pentium	40	200
Pentium	100	200
AMD	40	200
AMD	100	200
AMD	40	200
AMD	100	200
Pentium	40	200
AMD	40	200
AMD	100	200
AMD	100	300
AMD	200	500
Pentium	60	200
Pentium	20	200
Pentium	40	200
Pentium	100	200

Correlated Sub-Queries

This term refers to executing a Sub-query that depends on values in the Outer query.

Consider two data sets that contain the following information.

Table A			Table B		
Code	Name	Date	Code	Height	Weight
aa	Jimmy	24dec92	aa	66	150
bb	Jack	12oct91	ee	78	220
cc	Phil	03aug65	bb	74	210
dd	Sam	14feb78	ff	70	175

Example 1: Select everyone from Table A greater than 72 inches in height.

```

Command ==>
00001 data a;
00002 input code $
00003 name $
00004 date : date7.;
00005 cards;
00006 aa Jimmy 24dec92
00007 bb Jack 12oct91
00008 cc Phil 03aug65
00009 dd Sam 14feb78
00010 run;
00011
00012 data b;
00013 input code $
00014 height
00015 weight;
00016 cards;
00017 aa 66 150
00018 ee 78 220
00019 bb 74 210
00020 ff 70 175
00021 run;
00022
00023 title 'Inner Query';
00024
00025 proc sql;
00026 select height
00027 from a,b
00028 where a.code=b.code;
00029 quit;
00030
00031 title 'Total Query - Select Everyone from Table A Greater than 72 inches
00032 in Height';
00033 proc sql;
00034 select name,
00035 date format date7,
00036 from a where 72 < (select height from b where a.code=b.code);
00037 quit;

```

Process Flow:

Row 1:			
Sub-query:	a.code=aa	b.code=aa	Match
Outer-query:	Height 72	< 66?	No
Row 2:			
Sub-query:	a.code=bb	b.code=bb	Match
Outer-query:	Height 72	Height 72	Yes
Row 3:			
Sub-query:	a.code=cc	b.code=	No match
Outer-query:			Does not take place
Row 4:			
Sub-query:	a.code=dd	b.code=	No match
Outer-query:			Does not take place

Example 2: Select the most profitable computers where the average quality rating for equipment suppliers is greater than 6.5.

Two tables contain different sets of information for a computer company.

In Saved.Compsupp quality ratings are kept on the suppliers of computer equipment pertaining to delivery targets, product quality and technical support.

```

Command ==>
00001 proc sql;
00002 select * from saved.compsupp;
00003 quit;

```

	DELIVERY	PRODUCT	SUPPORT
FLOPPY COMPUTERS	8	4	5
KETCHUP COMPUTERS	7	7	9

Ratings are based on a 10-point scale (1=poor, 10=excellent)

In Saved.Computer information is kept on the supplier, CPU type, retail price and wholesale price of manufactured computers.

The following will analyze which CPU type is the most profitable and take into account the current quality ratings of our suppliers.

The Outer query provides statistics on the most profitable CPUs, while the Inner query chooses only those suppliers whose average overall rating is above 6.5.

```

Command ==>
00001 proc sql;
00002 select supplier label='Supplier',
00003 cpu,
00004 avg(retail-wholesale)
00005 label='Average Profit' format=7.2,
00006 std(retail-wholesale)
00007 label='Standard Deviation' format=8.2
00008 from saved.computer
00009 where 6.5 <
00010 (select mean(delivery,product,support)
00011 from saved.compsupp
00012 where computer.supplier=compsupp.supplier)
00013 group by supplier,cpu;
00014 quit;

```

	CPU	Average Profit	Standard Deviation
KETCHUP COMPUTERS	AMD	237.50	77.24
KETCHUP COMPUTERS	Pentium	187.50	29.00
KETCHUP COMPUTERS	Zeon	275.00	85.74

With conventional processing, a multi-step process is required.

```

Command ==>
00001 data ckeupp;
00002 select * from saved.compsupp;
00003 if mean(delivery,product,support) > 6.5;
00004 run;
00005 proc sort data=ckeupp;
00006 by supplier;
00007 run;
00008 proc sort data=saved.computer out=sorted;
00009 by supplier;
00010 run;
00011 data both;
00012 merge ckeupp(in=lnok) sorted;
00013 by supplier;
00014 if lnok;
00015 profit = retail - wholesale;
00016 run;
00017 proc sort data=both;
00018 by supplier cpu;
00019 run;
00020 proc means data=both noprint;
00021 var profit;
00022 by supplier cpu;
00023 output out = stata(drop=_type_ _freq_)
00024 mean = supprofit std = stprofit;
00025 run;
00026 proc print data=stata split='';
00027 label supprofit='Average Profit'
00028 stprofit='Standard Deviation';
00029 format supprofit 7.2
00030 stprofit 8.2;
00031 run;

```

Doc	SUPPLIER	CPU	Average Price	Standard Deviation
1	KETCHUP COMPUTERS	AMD	247.50	77.24
2	KETCHUP COMPUTERS	Pentium	187.50	25.00
3	KETCHUP COMPUTERS	Zeon	275.00	95.74

Syntax Summary

SAS Online Help information

Syntax

This subsection begins describing the syntax of the SQL procedure and its options, statements, and language components.

```
PROC SQL <option-list>;
  ALTER TABLE alter-statement;
  CREATE create-statement;
  DELETE delete-statement;
  DESCRIBE describe-statement;
  DROP drop-statement;
  INSERT insert-statement;
  RESET <reset statement option-list>;
  SELECT select-statement;
  UPDATE update-statement;
  VALIDATE validate-statement;
```

Select Statement Options

SELECT Statement

```
SELECT <DISTINCT> object-item <object item>...
  <INTO :macro-variable-specification <:macro-variable-specification>...>
  FROM from-list
  <WHERE sql-expression>
  <GROUP BY group-by-item <group-by-item>...>
  <HAVING sql-expression>
  <ORDER BY order-by-item <order-by-item> ...>;
```

For more information, refer to SAS Online Help.

General Form and Language Components

Select Distinct
Item

Case Structure ...

As ...

Format= ...

Label= ...

Outer Union ...

Intersect ...

Except...

item

item Expression

Function(item)

Into :host variable

:host variable

From SAS file As ...

Full Join

Left Join

Right Join

SAS file As ...

On key=key

(rows) Where

clause Not

Between x And y

Exists (subquery)

As ...

In ...

Like ...

Is ...

Group By

items

(groups) Having

clause

(as Where clause)

Order by

items

;

Order of Options

Select statement options are placed in the following order, separated by blanks.

```
Command ==>
00001 Proc sql;
00002   select items                /* required */
00003   from data file or view      /* required */
00004   where clause                 /* optional */
00005   group by item,item,...      /* optional */
00006   having clause              /* optional */
00007   order by item,item,...     /* optional */
00008 ;
00009 quit;
```

Notes

- items separated by commas, for example: disk, type, CPU.
- data file or view can be one or more SAS data file or views, for example: Saved.Computer, Saved.Compsupp.
- clause can be simple or may contain nesting.

Summary Functions

Some functions have more than one name to satisfy SAS and SQL conventions. Some of these are listed below.

Name	Function
AVG, MEAN	Mean of values
COUNT, FREQ, N	Number of non-missing values
CSS	Corrected sum of squares
CV	Coefficient of variation (percent)
MAX	Maximum value
MIN	Minimum value
NMISS	Number of missing values
PRT	Probability of a greater absolute value
RANGE	Range of values
STD	Standard deviation
STDERR	Standard error of the mean
SUM	Sum of values
SUMWGT	Sum of the WEIGHT variable values (*)
T	Student's value for testing that the population mean is zero
USS	Uncorrected sum of squares
VAR	Variance

(*) Currently there is no way to specify a Weight variable with SQL. All rows have a weighting of one.