Producing Readable Output with SQL*Plus

Chapter 8

Objectives

After completing this lesson, you should be able to do the following:

- **Produce queries that require an input variable**
- **Customize the SQL*Plus environment**
- **Produce more readable output**
- **Create and execute script files**
- **Save customizations**

Lesson Aim

In this lesson, you will leam how to include SQL*Plus commands to produce more readable SQL output.

You can create a command file containing a WHERE clause to restrict the rows displayed. To change the condition each time the command file is m n. you use substitution variable. Substitution variables can replace values in the WHERE clause, a text string, and even a column or a table name.

Substitution Variables

- Use SQL*Plus substitution variables to temporarily store values.
  - Single ampersand (&)
  - Double ampersand (&&)
  - DEFINE and ACCEPT commands
- Pass variable values betvveen SOL statements.
- Dynamically alter headers and footers.

Using the & Substitution Variable

Use a variable prefixed with an ampersand (&) to prompt the user for a value.

```
SELECT  empno, ename, sal, deptno
FROM    emp
WHERE   empno =  &employee_num ;
```

Enter value for employee_num:

old 3: WHERE empno = &employee_num
new 3: WHERE empno = 7369

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20

Single-Ampersand Substitution Variable

When running a report, users often want to restrict the data returned dynamically. SQL*Plus provides this flexibility by means of user variables. Use an ampersand (&) to identify each variable in your SQL statement. You do not need to define the value of each variable.

Using the SET VERIFY Command

Toggling the display of the text of a command before and after SQL*Plus replaces substitution variables with values.

```
SET VERiFY ON
SELECT  empno, ename, sal, deptno
FROM    emp
WHERE       empno  =  &employee_num;
```

Enter value for employee_num:

```
old 3: WHERE empno = &employee_num
new 3: WHERE empno = 7369
```

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20

The SET VERIFY Command

To confirm the changes in the SQL statement, use the SQL*Plus SET VERIFY command. Setting SETVERIFY ON forces SQL*Plus to display the text of a command before and after it replaces substitution variables with values.

Character and Date Values with Substitution Variables

Use single quotation marks for date and character values.

```
SELECT ename, deptno, sal*12
FROM   emp
WHERE  job = &job_title ;
```

```
Enter value for job_title:

old 3: WHERE job = &job_title
new 3: WHERE job = 'ANALYST'
```

ENAME	DEPTNO	SAL*12
SCOTT	20	36000
FORD	20	36000

In a WHERE clause, date and character values must be enclosed within single quotation marks. The same rule applies to the substitution variables.

Specifying Character and Date Values with Substitution Variables

To avoid entering quotation marks at runtime, enclose the variabie in single quotation marks within the SQL statement itself.

```
SELECT ename, deptno, sal*12
FROM   emp
WHERE  job = '&job_title' ;
```

Enter value for job_title: **ANALYST**

old 3: WHERE job = '&job_title'

new 3: WHERE job = 'ANALYST'

ENAME	DEPTNO	SAL*12
SCOTT	20	36000
FORD	20	36000

The slide shows a query to retreive the employee name, department number, and annual salary of all employees based on the job title entered at the prompt by the user.

Using Functions with Substitution Variables (1)

You can also use functions such as UPPER and LOWER with the ampersand.

```
SELECT ename, deptno, sal*12
FROM   emp
WHERE  job = UPPER(&job_title);
```

```
Enter value for job_title:  'anaLYst'

old 3: WHERE job = UPPER(&job_title)
new 3: WHERE job = UPPER('anaLYst')
```

ENAME	DEPTNO	SAL*12
SCOTT	20	36000
FORD	20	36000

Use **UPPER ('&job_title')** so that the user does not have to enter the job title in uppercase.

Using Functions with Substitution Variables (2)

Substitution variables may also be enclosed with single quotation marks inside the UPPER and LOWER functions.

```
SELECT ename, deptno, sal*12
FROM   emp
WHERE  job = UPPER('&job_title');
```

```
Enter value for job_title:  analYsT

old 3: WHERE job = UPPER('&job_title')
new 3: WHERE job = UPPER('analYsT')
```

ENAME	DEPTNO	SAL*12
SCOTT	20	36000
FORD	20	36000

The slide example displays the use of **UPPER ('&job_title').** In this case the user does not have to enter the job title in uppercase.

Specifying Column Names, Expressions, and Text at Runtime

Use substitution variables to supplement the following:

- **WHERE condition**
- **ORDER BY clause**
- **Column expression**
- **Table name**
- **Entire SELECTstatement**

Specifying Column Names, Expressions, and Text at Runtime

Not only can you use the substitution variable, but the WHERE clause of a SQL statement, but also these variables can be used to substitute column names, expressions, or text.

Example

Display the employee number and any other column and any condition of employees.

```
SELECT      empno,   &column_name
FROM     emp
WHERE       &condition;

Enter value for column_name:      job

Enter value for condition:      deptno=10
```

old 3: WHERE &condition
new 3: WHERE deptno=10

EMPNO	JOB
7782	MANAGER
7839	PRESIDENT
7934	CLERK

Specifying Column Names,

Expressions, and Text at Runtime

```
SELECTempno,  ename,   &column_name
FROM      emp
WHERE &condition
ORDER BY &order_column ;
```

```
Enter value for column_name        : sal
Enter value for condition          : sal >= 3000
Enter value for order aolumn       : ename
```

```
old 4: ORDER BY &order_column
new 4: ORDER BY ename
```

EMPNO	ENAME	JOB	SAL
7902	FORD	ANALYST	3000
7839	KING	PRESIDENT	5000
7788	SCOTT	ANALYST	3000

Specifying Column Names, Expressions, and Text at Runtime (continued)

The slide example displays the employee number, name, job title, and any other column specified by the user at runtime, from the EMP table. The user can also specify the condition for retrieval of rows and the column name by which the resultant data has to be ordered.

Using the && Substitution Variable

Use the double-ampersand (&&) if you want to reuse the variable value without prompting the user each time.

```
SELECT  empno,  ename,  job,  &&column_name
FROM  emp
ORDER BY  &column_name ;
```

```
Enter value for column_name:     deptno
```

```
old 1: SELECT empno, ename, job, &&column_name
new 1: SELECT empno, ename, job, deptno
old 3: ORDER BY &column_name
new 3: ORDER BY deptno
```

EMPNO	ENAME	JOB	DEPTNO
7782	CLARK	MANAGER	10
7839	KING	PRESIDENT	10

14 rows selected.

Double-Ampersand Substitution Variable

You can use the double-ampersand (&&) substitution variable if you want to reuse the variable value without prompting the user each time. The user will see the prompt for the value only once. In the example on the slide, the user is asked to give the value for variable column name only once. The value supplied by the user (deptno) is used both for display and ordering of data.

SQL*Plus stores the value supplied by using the DEFINE command: it will use it again whenever you reference the variable name. Once a user variable is in place, you need to use the UNDEFINE command to delete it.

```
SELECT  empno,  ename,  job,  &&column_name
FROM  emp
ORDER BY  &name ;
```

old 3: ORDER BY &name
new 3: ORDER BY ename

EMPNO	ENAME	JOB	SAL
7876	ADAMS	CLERK	1100
7499	ALLEN	SALESMAN	1600
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7902	FORD	ANALYST	3000
7900	JAMES	CLERK	950
7566	JONES	MANAGER	2975
7839	KING	PRESIDENT	5000
7654	MARTIN	SALESMAN	1250
7934	MILLER	CLERK	1300
7788	SCOTT	ANALYST	3000
7369	SMITH	CLERK	800
7844	TURNER	SALESMAN	1500
7521	WARD	SALESMAN	1250

14 rows selected.

Defining User Variables

* **You can predefine variables using one of two SQL*Plus commands:**
  - **DEFINE: Create a CHAR datatype user variable**
  - **ACCEPT: Read user input and store it in a variable**
- **If you need to predefine a variable that includes spaces, you must enclose the value within single quotation marks when using the DEFINE command.**

Defining User Variables

You can predefine user variables before executing a SELECT statement. SQL*Plus provides two commands for defining and setting user variables: DEFINE and ACCEPT.

Commad	Description
DEFINE *variable = value*	Creates a CHAR datatype user variable and assigns a value to it
DEFINE variable	Displays the variable, its value, and its datatype
DEFINE	Displays all user variables with value and datatype
ACCEPT (see syntax on next slide)	Reads aline of user input and stores it in a variable

The ACCEPT Command

- **Creates a customized prompt when accepting user input**
- **Explicitly defines a NUMBER or DATE datatype variable**
- **Hides user input for security reasons**

ACCEPT variable [datatype] [FORMAT format] [PROMPT text] [HIDE]

The ACCEPT Command
In The syntax:

variable	is the name of the variable that stores the value (If it does not exist, SQL*Plus creates it.)
datatype	is NUMBER, CHAR, or DATE (CHAR has a maximum length limit of 240 bytes. DATE checks against a format model, and the data type is CHAR.)
FOR[MAT] format	specifies the format model – for example, A10 or 9.999
PROMPT text	displays the text before the user can enter the value.
HIDE	suppresses what the user enters —for example, a password

Note: Do not prefix the SQL*Plus substitution parameter with the ampersand (&) when referencing the substitution parameter in the ACCEPT command.

Using the ACCEPT Command

```
ACCEPT      dept  PROMPT  ‘ Provide the department name:  ‘
SELECT      *
FROM dept
WHERE       dname = UPPER( ‘&dept’ ) ;
```

Provide the department name: **sales**

old 3: WHERE dname= UPPER('&dept')
new 3: WHERE dname= UPPER('sales')

DEPTNO	DNAME	LOC
30	SALES	CHICAGO

Using the ACCEPT Command

The ACCEPT command reads in a variable named DEPT. The prompt that it displays when asking the user for the variable is ‘ Provide the department name: ‘. The SELECT statement then takes the department value that the user enters and uses it to retrieve the appropriate row from the DEPT table.

If the user enters a valid value for department name, the SELECT statemsnt executes in the same way as any other SELECT statement, taking the user-entered value and using it in the WHERE clause to compare with DNAME.

Note that the & character does not appear with the DEPT variable in the ACCEPT command. The & appears only in the SELECT statement.

Using the DEFINE Command

```
DEFINE    occupation=clerk
SELECT    *
FROM      emp
WHERE     job = UPPER( '&occupation' )
```

```
old 3: WHERE job = UPPER( '&occupation' )
new 3: WHERE job = UPPER( 'clerk' )
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/1980	800		20
7876	ADAMS	CLERK	7788	12/01/1983	1100		20
7900	JAMES	CLERK	7698	03/12/1981	950		30
7934	MILLER	CLERK	7782	23/01/1982	1300		10

Guidelines

The ACCEPT and DEFINE commands will create a variable if the variable does not exist; these commands will automatically redefine a variable if it exists.

When using the DEFINE command, use single quotation marks (' ') to enclose a string that contains an embedded space.

DEFINE and UNDEFINE Commands

- **A variable remains defined until you either:**
  - **Use the UNDEFINE command to clear it**
  - **Exit SQL*Plus**
- **You can verify your changes with the DEFINE command.**
- **To define variables for every session, modify your *login.sql file* so that the variables are created at startup.**

The DEFINE and UNDEFINE Commands

Variables are defined until you either:

- Issue the UNDEFINE command on a varible
- Exit SQL*Plus

When you undefine variables, you can verify your changes with the DEFINE command. When you exit SQL*Plus, variables defined during that session are lost. To define those variables for every session, modify your *login.sql* file so that those variables are created at startup.

Using the DEFINE Command

Create a variable to hold the department name.

```
SQL> DEFINE deptname = sales
SQL> DEFINE deptname
```

```
DEFINE DEPTNAME       = "sales" (CHAR)
```

- **Use the variable as you would any other variable.**

```
SELECT   *
    FROM       dept
    WHERE    dname  =  UPPER('&deptname') ;
```

```
old 3: WHERE dname = UPPER('&deptname')
new 3: WHERE dname = UPPER('sales')
```

DEPTNO	DNAME	LOC
30	SALES	CHICAGO

Using the DEFINE Command

You can *use* the DEFINE cornmand to create a variable and then use the variables as you would any other variable. The example on the slide creates a variable DEPTNAME that contains the department name SALES. The SQL statement then uses this variable to display the number and location of the sales department.

```
UNDEFINE   deptname

DEFINE   deptname

symbol deptname is   UNDEFINED
```

Customizing the SQL*Plus Environment

- **Use SET commands to control current session.**

```
SET  system variable  value
```

Verify what you have set by using the SHOW command.

```
SQL> SET  ECHO   ON
```

```
SQL>  SHOW  ECHO
echo  ON
```

Customizing the SOL*Plus Environment

You can control the environment in which SQL*Plus is currently operating by using the SET commands.

In the syntax:

System_variable	is a variable that controls one aspect of the session environment
value	is a value for the system variable

You can verify what you have set by using the SHOW command. The SHOW command on the slide checks whether ECHO had been set on or off.

To see all SET variable values, use the SHOW ALL command.

SET Command Variables

- **ARRAYSIZE {20 | *n}***
- **COLSEP {_ | *text}***
- **FEEDBACK {6 | *n* |OFF | ON}**
- **HEAD1NG (OFF I ON>**
- **LINESIZE {80 | n}**
- **LONG {80 | *n}***
- **PAGESIZE {24 | *n}***
- **PAUSE (OFF | ON | *text}***
- **TERMOUT {OFF | ON}**

Saving Customizations in the *login.sql* File

The *login.sql file* contains standard SET and other SQL*Plus commands that are implemented at login.

You can modify *login.sql to* contain additional SET commands.

Default Settings Using the *login.sql File*

The *login.sql* file contains standard SET and other SQL*Plus commands that yon may require for every session. The file is read and commands are implemented at login. When you log out of your session, all customized settings are lost.

Changing the Default Settings

The settings implemented by *login.sql* can be changed during the current session. Changes made are current only for that session. As soon as you log out, those settings are lost.

Add permanent changes to settings to the *login.sql* file.

SQL*Plus Format Commands

COLUMN ***[column option]***

TTITLE [texf|OFF|ON]

BTITLE [texf|OFF|ON]

BREAK [ON ***report_elemenf]***

The COLUMN Command

Controls display of a column

COL[UMN] [{column | alias} *[option]]*

- **CLE[AR]: Clears any column formats**
- **FOR[MAT] *format* Changes the display of the column using a format model**
- **HEA[DING] *text* Sets the column heading**
- **JUS[TIFY] *{align}:* Aligns the column heading to be left, center, or right**

Using the COLUMN Command

Createcolumn headings.

```
COLUMN  ename  HEADING  'Employee | Name'  FORMAT  A15

COLUMN sal  JUSTIFY LEFT FORMAT $99,990.00

COLUMN mgr FORMAT 999999999 NULL 'No manager'
```

Display the current setting for the ENAME column.

```
COLUMN  ename
```

Clear settings for the ENAME column.

```
COLUMN  ename   CLEAR
```

COLUMN Format Models

Element	Description	Example	Result
A*n*	Sets a display width of *n*	N/A	N/A
9	Single zero-suppression digit	999999	1234
0	Enforces leading zero	099999	01234
$	Floating dollar sign	$9999	$1234
L	Local currency	L9999	L1234
.	Position of decimal point	9999.99	1234.00
,	Thousand separator	9,999	1,234

COLUMN **Format** Models

The slide displays sample COLUMN fomat models.

The Oracle Server displays a string of pound sıgns (#) in place of a whole number whose digits exceed the number of digils provided in the format model. It also displays pound signs in place of a value whose format model is alphanumeric but whose actual value is numeric.

Using the BREAK Command

Suppresses duplicates and sections rows

- **To suppress duplicates**

SQL> BREAK ON ename ON job

- **To section out rows at break values**

SQL> BREAK ON ename SKIP 4 ON job SKIP2

Using the TTITLE and BTITLE Commands

Display headers and footers.

TTI[TLE] [*text* \OFF|ON]

- **Set the report header.**

```
SQL> TTITLE 'Salary | Report'
```

- **Set the report footer.**

```
SQL> BTITLE 'Confidential'
```

Creating a Script File to Run a Report

1.Create the SOL SELECT statement.

2.**Save the SELECT statement to a script file.**

3.**Load the script file into an editör.**

4.Add formatting commands before the SELECT statement.

5.**Verify that the termination character followvs the SELECT statement.**

Creating a Script File to Run a Report

You can either enter each of the SQL*Plus commands at the SQL prompt or put all the commands, includmg the SELECT statement, in a command (or script) file. A typical script consists of at least one SELECT statement and several SQL*Plus commands.

How to Create a Script File

1. Create the SQL SELECT statement at the SQL prompt. Ensure that the data required for the report is accurate before you save the statement to a file and apply formatting commands. Ensure that the relevant ORDER BY clause is included if you intend to use breaks

2. Save the SELECT statement to a script file.

3. Edit the script file to enter the SQL*Plus commands.

4. Add the required formatting commands before the SELECT statement. Be certain not to place SQL*Plus commands within the SELECT statement.

5. Verify that the SELECT statement is followed by a run character, either a semicolon (;) or a Slash (/).

Creating a Script File to Run a Report

6. **Clear formatting commands after the SELECTstatement.**
7. **Save the script file.**
8. **Enter "START *filename*" to run the script.**

How to Create a Script File (continued)

6 Add the format-clearing SQL*Plus commands after the run character. As an alternative, you can call a reset file that contains all the format-clearing commands.

7 Save the script file with your changes.

8 In SQL*Plus, run the script file by entering START *file name o*r *@filename.* This command is required to read and execute the script file.

Guidelines

Yon can include blank lines between SQL*Plus commands in a script.

You can abbreviate SQL*Plus commands.

Include reset commands at the end of the file to restore the original SQL*Plus environment.

Summary

- Use SQL*Plus substitution variables to temporarily store values.
- Use SET commands to control current SQL*Plus environment.
- Use the COLUMN command to control the display of a column.
- Use the BREAK command to suppress duplicates and section rows.
- Use TTITLE and BTITLE to display headers and footers.

Practice 8

2. The ACCEPT command is a SQL command. True/Falsc

3 Write ascript file to display the employee name, job, and hiredate for all employces who started between a given range. Concatenate the name and job toücther. sı;panued b}1 a space and comma. and label Ihe column Employees. Prompt the ıserfortlıe t\vo ranucs using tlıe ACCEPT command Uscîlıe fomıat MM/DD'YY Sa\etlıe scnpt file aspS(j3.s(./!.

Saving Customizations in the *login.sql* File

The *login.sql file* contains standard SET and other SQL*Plus commands that are implemented at login.

You can modify *login.sql to* contain additional SET commands.

The ACCEPT Command

- **Creates a customized prompt when accepting user input**
- **Explicitly defines a NUMBER ör DATE datatype variabfe**
- **Hides user input for security reasons**

ACCEPT *variable* [datatype] [FORMAT format] [PROMPT *text]* [HIDE]

Using the && Substitution Variable

Use the double-ampersand (&&) if you want to reuse the variable value vvithout prompting the user each time.