SQL*Plus® User's Guide and Reference

Release 8.0

Part No. A53717–01

ORACLE®

Enabling the Information Age

Preface

The *SQL*Plus* (pronounced "sequel plus") *User's Guide and Reference* introduces the SQL*Plus program and its uses. It also provides a detailed description of each SQL*Plus command.

Audience

This Guide addresses business and technical professionals who have a basic understanding of the SQL database language. If you do not have any familiarity with this database tool, you should refer to the *Oracle8 Server SQL Reference Manual.* If you plan to use the PL/SQL database language in conjunction with SQL*Plus, refer to the *PL/SQL User's Guide and Reference* for information on using PL/SQL.

How to Use this Guide Refer to the following tables for a list of topics covered by this Guide, a description of each topic, and the number of the chapter that covers the topic.

PART I
Understanding SQL*Plus

Topic	*Description*	*Chapter Number*
Introduction	Gives an overview of SQL*Plus, instructions on using this Guide, and information on what you need to run SQL*Plus.	1
Learning SQL*Plus Basics	Explains how to start SQL*Plus and enter and execute commands. You learn by following step-by-step examples using sample tables.	2
Manipulating Commands	Also through examples, helps you learn to edit commands, save them for later use, and write interactive commands.	3
Formatting Query Results	Explains how you can format columns, clarify your report with spacing and summary lines, define page dimensions and titles, and store and print query results. Also uses step-by-step examples.	4
Accessing Databases	Tells you how to connect to default and remote databases, and how to copy data between databases and between tables on the same database. Includes one example.	5

PART II Reference

Topic	Description	Chapter Number
Starting SQL*Plus and Getting Help	Explains how to access SQL*Plus from the operating system prompt, and how to access online help.	6
Command Reference	Gives you a SQL*Plus command summary and detailed descriptions of each SQL*Plus command in alphabetical order.	7
COPY Command Messages and Codes	Lists copy command error messages, their causes, and appropriate actions for error recovery.	Appendix A
Release 8.0 Enhancements	Describes enhancements to SQL*Plus in Release 8.0.	Appendix B
SQL*Plus Limits	Lists the maximum values for elements of SQL*Plus.	Appendix C
SQL Command List	Provides a list of major SQL commands and clauses.	Appendix D
Security	Explains how to restrict users' access to certain SQL*Plus and SQL commands.	Appendix E
SQL*Plus Commands from Earlier Releases	Provides information on SQL*Plus commands from earlier Releases.	Appendix F
Glossary	Defines technical terms associated with Oracle and SQL*Plus.	Glossary

Related Publications   Related documentation includes the following publications:

- *SQL*Plus Quick Reference*
- *PL/SQL User's Guide and Reference*
- *Oracle8 Server SQL Reference Manual*
- *Oracle8 Server Concepts Manual*
- *Oracle8 Server Administrator's Guide*
- *Oracle8 Server Backup and Recovery Guide*
- *Oracle8 Server Application Developer's Guide*
- *Oracle8 Server Distributed Systems Manual*
- *Oracle8 Server Replication Manual*

- *Oracle8 Server Utilities Guide*
- *Oracle8 Server Messages Manual*
- *Oracle8 Server Migration Guide*
- *Oracle8 Server Reference Manual*
- *Oracle8 Server Tuning Guide*
- *Oracle8 Parallel Server Concepts and Administration Manual*
- *Oracle Tools for UNIX Administrator's Reference Guide*
- *Programmer's Guide to the Oracle Call Interface*
- *Programmer's Guide to the Oracle Pro*COBOL Precompiler*
- *Programmer's Guide to the Oracle Pro*C/C++ Precompiler*
- Oracle installation and user's manual(s) provided for your operating system

Your Comments Are Welcome

Oracle Corporation values and appreciates your comments as an Oracle user and reader of the manuals. As we write, revise, and evaluate, your opinions are the most important input we receive. At the back of this manual is a Reader's Comment Form that we encourage you to use to tell us both what you like and what you dislike about this (or other) Oracle manuals. If the form is not at the end of this manual, or if you would like to contact us, please use the following addresses and phone numbers.

For documentation questions/comments, contact:

SQL*Plus Documentation Manager
Australian Product Development Center
Oracle Systems (Australia) Pty. Limited
324 St. Kilda Road
Melbourne VIC 3004
Australia
+61 3 9209 1600 (telephone)
+61 3 9690 0043 (fax)

For product questions/comments, contact:

SQL*Plus Product Manager
Australian Product Development Center
Oracle Systems (Australia) Pty. Limited
324 St. Kilda Road
Melbourne VIC 3004
Australia
+61 3 9209 1600 (telephone)
+61 3 9690 0043 (fax)

Contents

PART II **REFERENCE**

Appendix A   **COPY Command Messages and Codes**

Appendix B   **Release 8.0 Enhancements**

Appendix C   **SQL*Plus Limits**

Appendix D   **SQL Command List**

Appendix E   **Security**

                      Glossary

                      Index

Understanding
SQL*Plus

Introduction

This chapter introduces you to SQL*Plus, covering the following topics:

- overview of the SQL*Plus
- using this guide
- what you need to run SQL*Plus

Overview of SQL*Plus

You can use the SQL*Plus program in conjunction with the SQL database language and its procedural language extension, PL/SQL. The SQL database language allows you to store and retrieve data in Oracle. PL/SQL allows you to link several SQL commands through procedural logic.

SQL*Plus enables you to manipulate SQL commands and PL/SQL blocks, and to perform many additional tasks as well. Through SQL*Plus, you can

- enter, edit, store, retrieve, and run SQL commands and PL/SQL blocks
- format, perform calculations on, store, and print query results in the form of reports
- list column definitions for any table
- access and copy data between SQL databases
- send messages to and accept responses from an end user

Basic Concepts The following definitions explain concepts central to SQL*Plus:

command	An instruction you give SQL*Plus or Oracle.
block	A group of SQL and PL/SQL commands related to one another through procedural logic.
table	The basic unit of storage in Oracle.
query	A SQL command (specifically, a SQL SELECT command) that retrieves information from one or more tables.
query results	The data retrieved by a query.
report	Query results formatted by you through SQL*Plus commands.

Who Can Use SQL*Plus

The SQL*Plus, SQL, and PL/SQL command languages are powerful enough to serve the needs of users with some database experience, yet straightforward enough for new users who are just learning to work with Oracle.

The design of the SQL*Plus command language makes it easy to use. For example, to give a column labelled ENAME in the database the clearer heading "Employee", you might enter the following command:

```
COLUMN ENAME HEADING EMPLOYEE
```

Similarly, to list the column definitions for a table called EMP, you might enter this command:

```
DESCRIBE EMP
```

Other Ways of Working with Oracle

Oracle tools for Network Computing Architecture help developers to productively and economically build, manage and deploy high-performance and robust enterprise applications for Network Computing.

Designer/2000	a set of client/server design tools for database applications
Developer/2000	a set of client/server and Web development tools
Discoverer/2000	a set of end-user query tools
Programmer/2000	a set of 3GL programming language interfaces
Oracle ConText Option	an option to include full text storage and retrieval in databases
Oracle Spatial Data Option	an option to include multi-dimensional (spatial) data in databases
Oracle Mobile Agents	a tool for applications using mobile and/or detached clients
Oracle Web Application Server	a tool which enables database access through Web browsers and the Internet
Oracle Open Gateway Technology	a tool which enables access to data in non-Oracle databases
Oracle Media Objects	a development tool for object-oriented multimedia applications
Oracle InterOffice	an electronic messaging (Email), calendar, scheduling and document management system

Oracle Power Objects	a cross-platform tool for developing client/server applications against heterogeneous back-end data sources
Sedona	a set of tools that enable cartridges to be programmed in multiple languages
Object Database Designer	a tool for developers using Object Oriented 3GL applications of Oracle8

Using this Guide

This Guide gives you information on SQL*Plus that applies to all operating systems. Some aspects of SQL*Plus, however, differ on each operating system. Such operating system specific details are covered in the Oracle installation and user's manual(s) provided for your system. Use these operating system specific manuals in conjunction with the *SQL*Plus User's Guide and Reference.*

Throughout this Guide, examples showing how to enter commands use a common command syntax and a common set of sample tables. Both are described below. You will find the conventions for command syntax particularly useful when referring to the reference portion of this Guide.

Conventions for Command Syntax

The following two tables describe the notation and conventions for command syntax used in this Guide.

Feature	*Example*	*Explanation*
uppercase	`BTITLE`	Enter text exactly as spelled; it need not be in uppercase.
lowercase italics	`column`	A clause value; substitute an appropriate value.
words with specific meanings	`c`	A single character.
	`char`	A CHAR value—a literal in single quotes—or an expression with a CHAR value.
	`d` or `e`	A date or an expression with a DATE value.

expr	An unspecified expression.	
m or *n*	A number or an expression with a NUMBER value.	
text	A CHAR constant with or without single quotes.	
variable	A user variable (unless the text specifies another variable type).	

Table 1 – 1 Commands, Terms, and Clauses

Other words are explained where used if their meaning is not explained by context.

Feature	*Example*	*Explanation*
vertical bar	|	Separates alternative syntax elements that may be optional or mandatory.
brackets	[OFF|ON]	One or more optional items. If two items appear separated by |, enter one of the items separated by |. Do not enter the brackets or |.
braces	{OFF|ON}	A choice of mandatory items; enter one of the items separated by |. Do not enter the braces or |.
underlining	{<u>OFF</u>|ON}	A default value; if you enter nothing, SQL*Plus assumes the underlined value.
ellipsis	n...	Preceding item(s) may be repeated any number of times.

Table 1 – 2 Punctuation

Enter other punctuation marks (such as parentheses) where shown in the command syntax.

Sample Tables   Many of the concepts and operations in this Guide are illustrated by a set of sample tables. These tables contain personnel records for a fictitious company. As you complete the exercises in this Guide, imagine that you are the personnel director for this company.

The exercises make use of the information in two sample tables:

EMP   Contains information about the employees of the sample company.

DEPT   Contains information about the departments in the company.

Figure 1 – 1 and Figure 1 – 2 show the information in these tables.

```
EMPNO ENAME  JOB       MGR  HIREDATE     SAL    COMM   DEPTNO
----- -----  --------  ---- -----------  ------ ------ ------
 7369 SMITH  CLERK     7902 17-DEC-80      800            20
 7499 ALLEN  SALESMAN  7698 20-FEB-81     1600    300     30
 7521 WARD   SALESMAN  7698 22-FEB-81     1250    500     30
 7566 JONES  MANAGER   7839 02-APR-81     2975            20
 7654 MARTIN SALESMAN  7698 28-SEP-81     1250   1400     30
 7698 BLAKE  MANAGER   7839 01-MAY-81     2850            30
 7782 CLARK  MANAGER   7839 09-JUN-81     2450            30
 7788 SCOTT  ANALYST   7566 09-DEC-82     3000            20
 7839 KING   PRESIDENT      17-NOV-81     5000            10
 7844 TURNER SALESMAN  7698 08-SEP-81     1500      0     30
 7876 ADAMS  CLERK     7788 12-JAN-83     1100            20
 7900 JAMES  CLERK     7698 03-DEC-81      950            30
 7902 FORD   ANALYST   7566 03-DEC-81     3000            20
 7934 MILLER CLERK     7782 23-JAN-82     1300            10
```

Figure 1 – 1 EMP Table

```
DEPTNO    DNAME          LOC
--------- -------------- -----------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
       40 OPERATIONS     BOSTON
```

Figure 1 – 2 DEPT Table

What You Need to Run SQL*Plus

To run SQL*Plus, you need hardware, software, operating system specific information, a username and password, and access to one or more tables.

Hardware and Software

Oracle and SQL*Plus can run on many different kinds of computers. Your computer's operating system manages the computer's resources and mediates between the computer hardware and programs such as SQL*Plus. Different computers use different operating systems. For information about your computer's operating system, see the documentation provided with the computer.

Before you can begin using SQL*Plus, both Oracle and SQL*Plus must be installed on your computer. Note that in order to take full advantage of the enhancements in SQL*Plus Release 8.0, you must have Oracle8. For a list of SQL*Plus Release 8.0 enhancements, see Appendix B.

If you have multiple users on your computer, your organization should have a Database Administrator (called a DBA) who supervises the use of Oracle.

The DBA is responsible for installing Oracle and SQL*Plus on your system. If you are acting as DBA, see the instructions for installing Oracle and SQL*Plus in the Oracle installation and user's manual(s) provided for your operating system.

Information Specific to Your Operating System

A few aspects of Oracle and SQL*Plus differ from one type of host computer and operating system to another. These topics are discussed in the Oracle installation and user's manual(s), published in a separate version for each host computer and operating system that SQL*Plus supports.

Keep a copy of your Oracle installation and user's manual(s) available for reference as you work through this Guide. When necessary, this Guide will refer you to your installation and user's manual(s).

Username and Password

When you start SQL*Plus, you will need a *username* that identifies you as an authorized Oracle user and a *password* that proves you are the legitimate owner of your username. See the PASSWORD command in Chapter 7 for details on how to change your password. The demonstration username, SCOTT, and password, TIGER, may be set up on your system during the installation procedure. In this case, you can use the Oracle username SCOTT and password TIGER with the EMP and DEPT tables (Figure 1 – 1 and Figure 1 – 2).

Multi-User Systems	If several people share your computer's operating system, your DBA can set up your SQL*Plus username and password. You will also need a system username and password to gain admittance to the operating system. These may or may not be the same ones you use with SQL*Plus.
Single-User Systems	If only one person at a time uses your computer, you may be expected to perform the DBA's functions for yourself. In that case, you can use the Oracle username SCOTT and password TIGER. If you want to define your own username and password, see the *Oracle8 Server SQL Reference Manual.*
Access to Sample Tables	Each table in the database is "owned" by a particular user. You may wish to have your own copies of the sample tables to use as you try the examples in this Guide. To get your own copies of the tables, see your DBA or run the Oracle-supplied command file named DEMOBLD (you run this file from your operating system, not from SQL*Plus).
	When you have no more use for the sample tables, remove them by running another Oracle-supplied command file named DEMODROP. For instructions on how to run DEMOBLD and DEMODROP, see the Oracle installation and user's manual(s) provided for your operating system.

2

Learning SQL*Plus Basics

This chapter helps you learn the basics of using SQL*Plus, including the following topics:

- getting started
- entering and executing commands
- getting help

Read this chapter while sitting at your computer and try out the examples shown. Before beginning, make sure you have access to the sample tables described in Chapter 1.

Getting Started

To begin using SQL*Plus, you must first become familiar with the functions of several keys on your keyboard and understand how to start and leave SQL*Plus.

Using the Keyboard

Several keys on your keyboard have special meaning in SQL*Plus. Table 2 – 1 lists these keys.

See your Oracle installation and user's manual(s) for your operating system to learn which physical key performs each function on the keyboard commonly used with your host computer.

> **Note:** A SQL*Plus key may perform different functions when pressed in other products or the operating system.

Fill in each blank in Table 2 – 1 with the name of the corresponding keyboard key. Then locate each key on your keyboard.

*SQL*Plus Key Name*	*Keyboard Key Name*	*Function*
[Return]	_____	End a line of input.
[Backspace]	_____	Move cursor left one character to correct an error.
[Pause]	_____	Suspend program operation and display of output.
[Resume]	_____	Resume program operation and output [Pause].
[Cancel]	_____	Halt program operation; return to the SQL*Plus command prompt.
[Interrupt]	_____	Exit SQL*Plus and return to the host operating system.

Table 2 – 1 SQL*Plus Special Keys and their Functions

Starting SQL*Plus

Now that you have identified important keys on your keyboard, you are ready to start SQL*Plus.

Example 2–1
Starting SQL*Plus

This example shows you how to start SQL*Plus. Follow the steps shown.

1. Make sure that Oracle has been installed on your computer.

2. Turn on your computer (if it is off) and log on to the host operating system (if required). If you are already using your computer, you need not log off or reset it. Simply exit from the program you are using (if any).

    You should see one or more characters at the left side of the screen. This is the operating system's command prompt, which signals that the operating system is ready to accept a command. In this Guide the operating system's prompt will be represented by a dollar sign ($). Your computer's operating system prompt may be different.

3. Enter the command SQLPLUS and press [Return]. This is an operating system command that starts SQL*Plus.

    > **Note:** Some operating systems expect you to enter commands in lowercase letters. If your system expects lowercase, enter the SQLPLUS command in lowercase.

    ```
    $ SQLPLUS
    ```

    SQL*Plus displays its version number, the date, and copyright information, and prompts you for your username (the text displayed on your system may differ slightly):

    ```
    SQL*Plus: Release 8.0 - on Fri Jun 27 09:39:26 1997

    (c) Copyright 1997 Oracle Corporation.
    All rights reserved.

    Enter user-name:
    ```

4. Enter your username and press [Return]. SQL*Plus displays the prompt "Enter password:".

5. Enter your password and press [Return] again. For your protection, your password does not appear on the screen.

    The process of entering your username and password is called *logging in*. SQL*Plus displays the version of Oracle to which you connected and the versions of available tools such as PL/SQL.

Next, SQL*Plus displays the SQL*Plus command prompt:

```
SQL>
```

The command prompt indicates that SQL*Plus is ready to accept your commands.

If SQL*Plus does not start, you should see a message meant to help you correct the problem. For further information, refer to the *Oracle8 Server Messages* manual for Oracle messages, or to your operating system manual for system messages.

Shortcuts to Starting
SQL*Plus

When you start SQL*Plus, you can enter your username and password, separated by a slash (/), following the command SQLPLUS. For example, if your username is SCOTT and your password is TIGER, you can enter

```
$ SQLPLUS SCOTT/TIGER
```

and press [Return]. You can also arrange to log in to SQL*Plus automatically when you log on to your host operating system. See the Oracle installation and user's manual(s) provided for your operating system for details.

Leaving SQL*Plus

When you are done working with SQL*Plus and wish to return to the operating system, enter the EXIT command at the SQL*Plus command prompt.

**Example 2–2
Exiting SQL*Plus**

To leave SQL*Plus, enter the EXIT command at the SQL*Plus command prompt:

```
SQL> EXIT
```

SQL*Plus displays the version of Oracle from which you disconnected and the versions of tools available through SQL*Plus. After a moment you will see the operating system prompt.

Before continuing with this chapter, follow steps 3, 4, and 5 of Example 2–1 to start SQL*Plus again. Alternatively, log in using the shortcut shown under "Shortcuts to Starting SQL*Plus" above.

Entering and Executing Commands

Entering Commands

Your computer's cursor, or pointer (typically an underline, a rectangular block, or a slash), appears after the command prompt. The cursor indicates the place where the next character you type will appear on your screen.

To tell SQL*Plus what to do, simply type the command you wish to enter. Usually, you separate the words in a command from each other by a space or tab. You can use additional spaces or tabs between words, if you wish, to make your commands more readable.

> **Note:** You will see examples of spacing and indentation throughout this Guide. When you enter the commands in the exercises, you do not have to space them as shown, but you may find them clearer to read if you do.

You can enter commands in capitals or lowercase. For the sake of clarity, all table names, column names, and commands in this Guide appear in capital letters.

You can enter three kinds of commands at the command prompt:

- SQL commands, for working with information in the database
- PL/SQL blocks, also for working with information in the database
- SQL*Plus commands, for formatting query results, setting options, and editing and storing SQL commands and PL/SQL blocks

The manner in which you continue a command on additional lines, end a command, or execute a command differs depending on the type of command you wish to enter and run. Examples of how to run and execute these types of commands are found on the following pages.

Getting Help

To get online help for SQL*Plus commands, type HELP at the command prompt followed by the name of the command. For example:

```
SQL>HELP ACCEPT
```

If you get a response indicating that help is not available, consult your database administrator. For more details about the help system, see the HELP command in Chapter 7.

Executing Commands

After you enter the command and direct SQL*Plus to execute it, SQL*Plus processes the command and re-displays the command prompt, indicating that you can enter another command.

Running SQL Commands

The SQL command language enables you to manipulate data in the database. See your *Oracle8 Server SQL Reference Manual* for information on individual SQL commands.

Example 2–3 Entering a SQL Command

In this example, you will enter and execute a SQL command to display the employee number, name, job, and salary of each employee in the sample table EMP.

1. At the command prompt, enter the first line of the command:

```
SQL> SELECT EMPNO, ENAME, JOB, SAL
```

If you make a mistake, use [Backspace] to erase it and re-enter. When you are done, press [Return] to move to the next line.

2. SQL*Plus will display a "2", the prompt for the second line. Enter the second line of the command:

```
2  FROM EMP WHERE SAL < 2500;
```

The semicolon (;) means that this is the end of the command. Press [Return]. SQL*Plus processes the command and displays the results on the screen:

```
    EMPNO ENAME        JOB              SAL
---------- ------------ ---------- ----------
     7369 SMITH        CLERK            800
     7499 ALLEN        SALESMAN        1600
     7521 WARD         SALESMAN        1250
     7654 MARTIN       SALESMAN        1250
     7782 CLARK        MANAGER         2450
     7844 TURNER       SALESMAN        1500
     7876 ADAMS        CLERK           1100
     7900 JAMES        CLERK            800
     7934 MILLER       CLERK           1300

9 rows selected
SQL>
```

After displaying the results and the number of rows retrieved, SQL*Plus displays the command prompt again. If you made a mistake and therefore did not get the results shown above, simply re-enter the command.

The headings may be repeated in your output, depending on the setting of a system variable called PAGESIZE. Whether you see the message concerning the number of records retrieved depends on the setting of a system variable called FEEDBACK. You will learn more about system variables later in this chapter in the section "Variables

that Affect Running Commands". To save space, the number of records selected will not be shown in the rest of the examples in this Guide.

Understanding SQL Command Syntax

Just as spoken language has syntax rules that govern the way we assemble words into sentences, SQL*Plus has syntax rules that govern how you assemble words into commands. You must follow these rules if you want SQL*Plus to accept and execute your commands.

Dividing a SQL Command into Separate Lines   You can divide your SQL command into separate lines at any points you wish, as long as individual words are not split between lines. Thus, you can enter the query you entered in Example 2–3 on one line:

```
SQL> SELECT EMPNO, ENAME, JOB, SAL FROM EMP WHERE SAL < 2500;
```

You can also enter the query on several lines:

```
SQL> SELECT
  2   EMPNO, ENAME, JOB, SAL
  3   FROM EMP
  4   WHERE SAL < 2500;
```

In this Guide, you will find most SQL commands divided into clauses, one clause on each line. In Example 2–3, for instance, the SELECT and FROM clauses were placed on separate lines. Many people find this most convenient, but you may choose whatever line division makes your command most readable to you.

Ending a SQL Command   You can end a SQL command in one of three ways:

- with a semicolon (;)
- with a slash (/) on a line by itself
- with a blank line

A semicolon (;) tells SQL*Plus that you want to run the command. Type the semicolon at the end of the last line of the command, as shown in Example 2–3, and press [Return]. SQL*Plus will process the command and store it in the SQL buffer (see "The SQL Buffer" below for details). If you mistakenly press [Return] before typing the semicolon, SQL*Plus will prompt you with a line number for the next line of your command. Type the semicolon and press [Return] again to run the command.

> **Note:** You cannot enter a comment (/* */) on the same line after a semicolon.

A slash (/) on a line by itself also tells SQL*Plus that you wish to run the command. Press [Return] at the end of the last line of the command.

SQL*Plus prompts you with another line number. Type a slash and press [Return] again. SQL*Plus will execute the command and store it in the buffer (see "The SQL Buffer" below for details).

A blank line tells SQL*Plus that you have finished entering the command, but do not want to run it yet. Press [Return] at the end of the last line of the command. SQL*Plus prompts you with another line number.

Press [Return] again; SQL*Plus now prompts you with the SQL*Plus command prompt. SQL*Plus does not execute the command, but stores it in the SQL buffer (see "The SQL Buffer" below for details). If you subsequently enter another SQL command, SQL*Plus overwrites the previous command in the buffer.

Creating Stored Procedures   Stored procedures are PL/SQL functions, packages, or procedures. To create stored procedures, you use SQL CREATE commands. The following SQL CREATE commands are used to create stored procedures:

- CREATE FUNCTION
- CREATE PACKAGE
- CREATE PACKAGE BODY
- CREATE PROCEDURE
- CREATE TRIGGER
- CREATE TYPE

Entering any of these commands places you in PL/SQL mode, where you can enter your PL/SQL subprogram (see also "Running PL/SQL Blocks" in this chapter). When you are done typing your PL/SQL subprogram, enter a period (.) on a line by itself to terminate PL/SQL mode. To run the SQL command and create the stored procedure, you must enter RUN or slash (/). A semicolon (;) will not execute these CREATE commands.

When you use CREATE to create a stored procedure, a message appears if there are compilation errors. To view these errors, you use SHOW ERRORS. For example:

```
SQL> SHOW ERRORS PROCEDURE ASSIGNVL
```

See Chapter 7 for a description of the SHOW command.

To execute a PL/SQL statement that references a stored procedure, you can use the EXECUTE command. EXECUTE runs the PL/SQL statement that you enter immediately after the command. For example:

```
SQL> EXECUTE :ID := EMP_MANAGEMENT.GET_ID('BLAKE')
```

See Chapter 7 for a description of the EXECUTE command.

The SQL Buffer

The area where SQL*Plus stores your most recently entered SQL command or PL/SQL block is called the *SQL buffer*. The command or block remains there until you enter another. Thus, if you want to edit or re-run the current SQL command or PL/SQL block, you may do so without re-entering it. See Chapter 3 for details about editing or re-running a command or block stored in the buffer.

SQL*Plus does not store the semicolon or the slash you type to execute a command in the SQL buffer.

> **Note:** SQL*Plus commands are not stored in the SQL buffer.

Executing the Current SQL Command or PL/SQL Block from the Command Prompt

You can run (or re-run) the current SQL command or PL/SQL block by entering the RUN command or the slash (/) command at the command prompt. The RUN command lists the SQL command or PL/SQL block in the buffer before executing the command or block; the slash (/) command simply runs the SQL command or PL/SQL block.

Running PL/SQL Blocks

You can also use PL/SQL subprograms (called *blocks*) to manipulate data in the database. See your *PL/SQL User's Guide and Reference* for information on individual PL/SQL statements.

To enter a PL/SQL subprogram in SQL*Plus, you need to be in PL/SQL mode. You are placed in PL/SQL mode when

- You type DECLARE or BEGIN at the SQL*Plus command prompt. After you enter PL/SQL mode in this way, type the remainder of your PL/SQL subprogram.

- You type a SQL command (such as CREATE FUNCTION) that creates a stored procedure. After you enter PL/SQL mode in this way, type the stored procedure you want to create.

SQL*Plus treats PL/SQL subprograms in the same manner as SQL commands, except that a semicolon (;) or a blank line does not terminate and execute a block. Terminate PL/SQL subprograms by entering a period (.) by itself on a new line.

SQL*Plus stores the subprograms you enter at the SQL*Plus command prompt in the SQL buffer. Execute the current subprogram by issuing a RUN or slash (/) command. Likewise, to execute a SQL CREATE command that creates a stored procedure, you must also enter RUN or slash (/). A semicolon (;) will not execute these SQL commands as it does other SQL commands.

SQL*Plus sends the complete PL/SQL subprogram to Oracle for processing (as it does SQL commands). See your *PL/SQL User's Guide and Reference* for more information.

You might enter and execute a PL/SQL subprogram as follows:

```
SQL> DECLARE
  2      x    NUMBER := 100;
  3  BEGIN
  4      FOR i IN 1..10 LOOP
  5          IF MOD (i, 2) = 0 THEN    --i is even
  6              INSERT INTO temp VALUES (i, x, 'i is even');
  7          ELSE
  8              INSERT INTO temp VALUES (i, x, 'i is odd');
  9          END IF;
 10          x := x + 100;
 11      END LOOP;
 12  END;
 13  .
SQL> /

PL/SQL procedure successfully completed.
```

When you run a subprogram, the SQL commands within the subprogram may behave somewhat differently than they would outside the subprogram. See your *PL/SQL User's Guide and Reference* for detailed information on the PL/SQL language.

Running SQL*Plus Commands

You can use SQL*Plus commands to manipulate SQL commands and PL/SQL blocks and to format and print query results. SQL*Plus treats SQL*Plus commands differently than SQL commands or PL/SQL blocks. For information on individual SQL*Plus commands, refer to the Command Reference in Chapter 7.

To speed up command entry, you can abbreviate many SQL*Plus commands to one or a few letters. Abbreviations for some SQL*Plus commands are described along with the commands in Chapters 3, 4, and 5. For abbreviations of all SQL*Plus commands, refer to the command descriptions in Chapter 7.

Example 2–4
Entering a SQL*Plus
Command

This example shows how you might enter a SQL*Plus command to change the format used to display the column SAL of the sample table EMP.

1. On the command line, enter this SQL*Plus command:

```
SQL> COLUMN SAL FORMAT $99,999 HEADING SALARY
```

   If you make a mistake, use [Backspace] to erase it and re-enter. When you have entered the line, press [Return]. SQL*Plus notes the new format and displays the SQL*Plus command prompt again, ready for a new command.

2. Enter the RUN command to re-run the most recent query (from Example 2–3). SQL*Plus reprocesses the query and displays the results:

```
SQL> RUN
1  SELECT EMPNO, ENAME, JOB, SAL
2* FROM EMP WHERE SAL < 2500
    EMPNO ENAME         JOB         SALARY
    ------- ------------- ---------- --------
     7369 SMITH         CLERK          $800
     7499 ALLEN         SALESMAN     $1,600
     7521 WARD          SALESMAN     $1,250
     7654 MARTIN        SALESMAN     $1,250
     7782 CLARK         MANAGER      $2,450
     7844 TURNER        SALESMAN     $1,500
     7876 ADAMS         CLERK        $1,100
     7900 JAMES         CLERK          $800
     7934 MILLER        CLERK        $1,300
```

   The COLUMN command formatted the column SAL with a dollar sign ($) and a comma (,) and gave it a new heading. The RUN command then re-ran the query of Example 2–3, which was stored in the buffer. SQL*Plus does not store SQL*Plus commands in the SQL buffer.

Understanding SQL*Plus Command Syntax

SQL*Plus commands have a different syntax from SQL commands or PL/SQL blocks.

Continuing a Long SQL*Plus Command on Additional Lines   You can continue a long SQL*Plus command by typing a hyphen at the end of the line and pressing [Return]. If you wish, you can type a space before typing the hyphen. SQL*Plus displays a right angle-bracket (>) as a prompt for each additional line. For example:

```
SQL> COLUMN SAL FORMAT $99,999 –
>   HEADING SALARY
```

Ending a SQL*Plus Command   You do not need to end a SQL*Plus command with a semicolon. When you finish entering the command, you can just press [Return]. If you wish, however, you can enter a semicolon at the end of a SQL*Plus command.

Variables that Affect Running Commands

The SQL*Plus command SET controls many variables—called *system variables*—the settings of which affect the way SQL*Plus runs your commands. System variables control a variety of conditions within SQL*Plus, including default column widths for your output, whether SQL*Plus displays the number of records selected by a command, and your page size. System variables are also called *SET command variables.*

The examples in this Guide are based on running SQL*Plus with the system variables at their default settings. Depending on the settings of your system variables, your output may appear slightly different than the output shown in the examples. (Your settings might differ from the default settings if you have a SQL*Plus LOGIN file on your computer.)

For more information on system variables and their default settings, see the SET command in Chapter 7. For details on the SQL*Plus LOGIN file, refer to the section "Setting Up Your SQL*Plus Environment" under "Saving Commands for Later Use" in Chapter 3 and to the SQLPLUS command in Chapter 6.

To list the current setting of a SET command variable, enter SHOW followed by the variable name at the command prompt. See the SHOW command in Chapter 7 for information on other items you can list with SHOW.

Saving Changes to the Database Automatically

Through the SQL DML commands UPDATE, INSERT, and DELETE—which can be used independently or within a PL/SQL block—specify changes you wish to make to the information stored in the database. These changes are not made permanent until you enter a SQL COMMIT command or a SQL DCL or DDL command (such as CREATE TABLE), or use the autocommit feature. The SQL*Plus autocommit feature causes pending changes to be committed after a specified number of successful SQL DML transactions. (A SQL DML transaction is either an UPDATE, INSERT, or DELETE command, or a PL/SQL block.)

You control the autocommit feature with the SQL*Plus SET command's AUTOCOMMIT variable. It has these four forms:

`SET AUTOCOMMIT ON`   Turns autocommit on.

`SET AUTOCOMMIT OFF`   Turns autocommit off (the default).

`SET AUTOCOMMIT n`	Commits changes after *n* SQL commands or PL/SQL blocks.
`SET AUTOCOMMIT IMMEDIATE`	Turns autocommit on.

Example 2–5 Turning Autocommit On

To turn the autocommit feature on, enter

```
SQL> SET AUTOCOMMIT ON
```

Alternatively, you can enter the following to turn the autocommit feature on:

```
SQL> SET AUTOCOMMIT IMMEDIATE
```

Until you change the setting of AUTOCOMMIT, SQL*Plus will automatically commit changes from each SQL command or PL/SQL block that specifies changes to the database. After each autocommit, SQL*Plus displays the following message:

```
commit complete
```

When the autocommit feature is turned on, you cannot roll back changes to the database.

To commit changes to the database after a number of SQL DML commands or PL/SQL blocks, for example, ten, enter

```
SQL> SET AUTOCOMMIT 10
```

SQL*Plus counts SQL DML commands and PL/SQL blocks as they are executed and commits the changes after the tenth SQL DML command or PL/SQL block.

> **Note:** For this feature, a PL/SQL block is regarded as one transaction, regardless of the actual number of SQL commands contained within it.

To turn the autocommit feature off again, enter the following command:

```
SQL> SET AUTOCOMMIT OFF
```

To confirm that AUTOCOMMIT is now set to OFF, enter the following SHOW command:

```
SQL> SHOW AUTOCOMMIT
autocommit OFF
```

For more information, see the AUTOCOMMIT variable of the SET command in Chapter 7.

Stopping a Command while it is Running

Suppose you have displayed the first page of a 50 page report and decide you do not need to see the rest of it. Press [Cancel]. (Refer to Table 2 – 1 at the beginning of this chapter to see how [Cancel] is labelled on your keyboard.) SQL*Plus will stop the display and return to the command prompt.

> **Note:** Pressing [Cancel] will not stop the printing of a file that you have sent to a printer with the OUT clause of the SQL*Plus SPOOL command. (You will learn about printing query results in Chapter 4.) You can stop the printing of a file through your operating system. For more information, see the installation and user's manual(s) provided for your operating system.

Collecting Timing Statistics on Commands You Run

Use the SQL*Plus command TIMING to collect and display data on the amount of computer resources used to run one or more commands or blocks. TIMING collects data for an elapsed period of time, saving the data on commands run during the period in a timer. See TIMING in Chapter 7 and the Oracle installation and user's manuals provided for your operating system for more information.

To delete all timers, enter CLEAR TIMING at the command prompt.

Running Host Operating System Commands

You can execute a host operating system command from the SQL*Plus command prompt. This is useful when you want to perform a task such as listing existing host operating system files.

To run a host operating system command, enter the SQL*Plus command HOST followed by the host operating system command. For example, this SQL*Plus command runs a host command, DIRECTORY *.SQL:

```
SQL> HOST DIRECTORY *.SQL
```

When the host command finishes running, the SQL*Plus command prompt appears again.

Getting Help

While you use SQL*Plus, you may find that you need to list column definitions for a table, or start and stop the display that scrolls by. You may also need to interpret error messages you receive when you enter a command incorrectly or when there is a problem with Oracle or SQL*Plus. The following sections describe how to get help for those situations.

Listing a Table Definition

To see the definitions of each column in a given table, use the SQL*Plus DESCRIBE command.

Example 2–6 **Using the DESCRIBE** **Command**	To list the column definitions of the three columns in the sample table DEPT, enter

```
SQL> DESCRIBE DEPT
```

The following output results:

```
Name                              Null?    Type
------------------------------- ------- -------------
DEPTNO                            NOT NULL NUMBER(2)
DNAME                                      VARCHAR2(14)
LOC                                        VARCHAR2(13)
```

> **Note:** DESCRIBE accesses information in the Oracle data dictionary. You can also use SQL SELECT commands to access this and other information in the database. See your *Oracle8 Server SQL Reference Manual* for details.

Listing PL/SQL Definitions

To see the definition of a function or procedure, use the SQL*Plus DESCRIBE command.

Example 2–7 **Using the DESCRIBE** **Command**	To list the definition of a function called AFUNC, enter

```
SQL> DESCRIBE afunc
```

The following output results:

```
FUNCTION afunc RETURNS NUMBER
Argument Name    Type     In/Out    Default?
--------------- -------- -------- ---------
F1               CHAR     IN
F2               NUMBER   IN
```

Controlling the Display

Suppose that you wish to stop and examine the contents of the screen while displaying a long report or the definition of a table with many columns. Press [Pause]. (Refer to Table 2 – 1 to see how [Pause] is labelled on your keyboard.) The display will pause while you examine it. To continue, press [Resume].

If you wish, you can use the PAUSE variable of the SQL*Plus SET command to have SQL*Plus pause after displaying each screen of a query or report. For more information, refer to the SET command in Chapter 7.

Interpreting Error Messages

If SQL*Plus detects an error in a command, it will try to help you out by displaying an error message.

Example 2–8
Interpreting an Error
Message

For example, if you misspell the name of a table while entering a command, an error message will tell you that the table or view does not exist:

```
SQL> DESCRIBE DPT
ERROR:
ORA-04043: object DPT does not exist
```

You will often be able to figure out how to correct the problem from the message alone. If you need further explanation, take one of the following steps to determine the cause of the problem and how to correct it:

- If the error is a numbered error for the SQL*Plus COPY command, look up the message in Appendix A of this Guide.

- If the error is a numbered error beginning with the letters "ORA", look up the message in the *Oracle8 Server Messages* manual or in the Oracle installation and user's manual(s) provided for your operating system to determine the cause of the problem and how to correct it.

- If the error is unnumbered, look up correct syntax for the command that generated the error in Chapter 7 of this Guide for a SQL*Plus command, in the *Oracle8 Server SQL Reference Manual* for a SQL command, or in the *PL/SQL User's Guide and Reference* for a PL/SQL block. Otherwise, contact your DBA.

3

Manipulating Commands

This chapter helps you learn to manipulate SQL*Plus commands, SQL commands, and PL/SQL blocks. It covers the following topics:

- editing commands
- saving commands for later use
- writing interactive commands
- using bind variables
- using REFCURSOR bind variables
- tracing statements

Read this chapter while sitting at your computer and try out the examples shown. Before beginning, make sure you have access to the sample tables described in Chapter 1.

Editing Commands

Because SQL*Plus does not store SQL*Plus commands in the buffer, you edit a SQL*Plus command entered directly to the command prompt by using [Backspace] or by re-entering the command.

You can use a number of SQL*Plus commands to edit the SQL command or PL/SQL block currently stored in the buffer. Alternatively, you can use a host operating system editor to edit the buffer contents.

Table 3 – 1 lists the SQL*Plus commands that allow you to examine or change the command in the buffer without re-entering the command.

Command	*Abbreviation*	*Purpose*
APPEND *text*	A *text*	adds *text* at the end of a line
CHANGE /old/new	C /old/new	changes *old* to *new* in a line
CHANGE /*text*	C /*text*	deletes *text* from a line
CLEAR BUFFER	CL BUFF	deletes all lines
DEL	(none)	deletes the current line
DEL *n*	(none)	deletes line *n*
DEL *	(none)	deletes the current line
DEL *n* *	(none)	deletes line *n* through the current line
DEL LAST	(none)	deletes the last line
DEL *m n*	(none)	deletes a range of lines (*m* to *n*)
DEL * n	(none)	deletes the current line through line *n*
INPUT	I	adds one or more lines
INPUT *text*	I *text*	adds a line consisting of *text*

LIST	L	lists all lines in the SQL buffer
LIST *n*	L *n* or *n*	lists line *n*
LIST *	L *	lists the current line
LIST *n* *	L *n* *	Lists line *n* through the current line
LIST LAST	L LAST	lists the last line
LIST *m n*	L *m n*	lists a range of lines (*m* to *n*)
LIST * *n*	L * *n*	Lists the current line through line *n*

Table 3 – 1 SQL*Plus Editing Commands

You will find these commands useful if you mistype a command or wish to modify a command you have entered.

Listing the Buffer Contents

Any editing command other than LIST and DEL affects only a single line in the buffer. This line is called the *current line.* It is marked with an asterisk when you list the current command or block.

Example 3–1
Listing the Buffer Contents

Suppose you want to list the current command. Use the LIST command as shown below. (If you have EXITed SQL*Plus or entered another SQL command or PL/SQL block since following the steps in Example 2–3, perform the steps in that example again before continuing.)

```
SQL> LIST
  1  SELECT EMPNO, ENAME, JOB, SAL
  2* FROM EMP WHERE SAL < 2500
```

Notice that the semicolon you entered at the end of the SELECT command is not listed. This semicolon is necessary to mark the end of the command when you enter it, but SQL*Plus does not store it in the SQL buffer. This makes editing more convenient, since it means you can add a new line to the end of the buffer without removing a semicolon from the line that was previously the last.

Editing the Current Line

The SQL*Plus CHANGE command allows you to edit the current line. Various actions determine which line is the current line:

- LIST a given line to make it the current line.

- When you LIST or RUN the command in the buffer, the last line of the command becomes the current line. (Using the slash (/) command to run the command in the buffer does not affect the current line, however.)

- If you get an error message, the line containing the error automatically becomes the current line.

Example 3–2 Making an Error in Command Entry

Suppose you try to select the DEPTNO column but mistakenly enter it as DPTNO. Enter the following command, purposely misspelling DEPTNO in the first line:

```
SQL> SELECT DPTNO, ENAME, SAL
  2 FROM EMP
  3 WHERE DEPTNO = 10;
```

You see this message on your screen:

```
SELECT DPTNO, ENAME, SAL
       *
ERROR at line 1:
ORA–0904:  invalid column name
```

Examine the error message; it indicates an invalid column name in line 1 of the query. The asterisk shows the point of error—the mistyped column DPTNO.

Instead of re-entering the entire command, you can correct the mistake by editing the command in the buffer. The line containing the error is now the current line. Use the CHANGE command to correct the mistake. This command has three parts, separated by slashes or any other non-alphanumeric character:

- the word CHANGE or the letter C

- the sequence of characters you want to change

- the replacement sequence of characters

The CHANGE command finds the first occurrence in the current line of the character sequence to be changed and changes it to the new sequence. If you wish to re-enter an entire line, you do not need to use the CHANGE command: re-enter the line by typing the line number followed by a space and the new text and pressing [Return].

Example 3–3
Correcting the Error

To change DPTNO to DEPTNO, change the line with the CHANGE command:

```
SQL> CHANGE /DPTNO/DEPTNO
```

The corrected line appears on your screen:

```
1* SELECT DEPTNO, ENAME, SAL
```

Now that you have corrected the error, you can use the RUN command to run the command again:

```
SQL> RUN
```

SQL*Plus lists the command, and then runs it:

```
1  SELECT  DEPTNO, ENAME, SAL
2  FROM    EMP
3* WHERE   DEPTNO = 10
 DEPTNO ENAME      SALARY
─────── ────────── ───────
     10 CLARK      $2,450
     10 KING       $5,000
     10 MILLER     $1,300
```

Note that the column SAL retains the format you gave it in Example 2–4. (If you have left SQL*Plus and started again since performing Example 2–4, the column has reverted to its original format.)

For information about the significance of case in a CHANGE command and on using wildcards to specify blocks of text in a CHANGE command, refer to CHANGE in Chapter 7.

Adding a New Line

To insert a new line after the current line, use the INPUT command.

To insert a line before line 1, enter a zero ("0") and follow the zero with text. SQL*Plus inserts the line at the beginning of the buffer and that line becomes line 1.

```
SQL> 0 SELECT EMPNO
```

Example 3–4
Adding a Line

Suppose you want to add a fourth line to the SQL command you modified in Example 3–3. Since line 3 is already the current line, enter INPUT (which may be abbreviated to I) and press [Return]. SQL*Plus prompts you for the new line:

```
SQL> INPUT
  4
```

Enter the new line. Then press [Return]. SQL*Plus prompts you again for a new line:

```
4   ORDER BY SAL
5
```

Press [Return] again to indicate that you will not enter any more lines, and then use RUN to verify and re-run the query.

Appending Text to a Line

To add text to the end of a line in the buffer, use the APPEND command:

1. Use the LIST command (or just the line number) to list the line you want to change.

2. Enter APPEND followed by the text you want to add. If the text you want to add begins with a blank, separate the word APPEND from the first character of the text by two blanks: one to separate APPEND from the text, and one to go into the buffer with the text.

**Example 3–5
Appending Text to a Line**

To append a space and the clause DESC to line 4 of the current query, first list line 4:

```
SQL> LIST 4
  4* ORDER BY SAL
```

Next, enter the following command (be sure to type two spaces between APPEND and DESC):

```
SQL> APPEND  DESC
  4* ORDER BY SAL DESC
```

Use RUN to verify and re-run the query.

Deleting Lines

To delete lines in the buffer, use the DEL command:

1. Use the LIST command (or just the line numbers) to list the lines you want to delete.

2. Enter DEL with an optional clause.

Suppose you want to delete the current line to the last line inclusive. Use the DEL command as shown below.

```
SQL> DEL * LAST
```

DEL makes the following line of the buffer (if any) the current line.

For more information, see the DELETE command in Chapter 7.

Editing Commands with a System Editor

Your host computer's operating system has one or more text editors that you can use to create and edit host system files. Text editors perform the same general functions as the SQL*Plus editing commands, but you may find them more familiar.

You can run your host operating system's default text editor without leaving SQL*Plus by entering the EDIT command:

```
SQL> EDIT
```

EDIT loads the contents of the buffer into your system's default text editor. You can then edit the text with the text editor's commands. When you tell the text editor to save edited text and then exit, the text is loaded back into the buffer.

To load the buffer contents into a text editor other than the default, use the SQL*Plus DEFINE command to define a variable, _EDITOR, to hold the name of the editor. For example, to define the editor to be used by EDIT as EDT, enter the following command:

```
SQL> DEFINE _EDITOR = EDT
```

You can also define the editor to be used by EDIT in your user or site profile. See "Setting Up Your SQL*Plus Environment" in Chapter 3 and DEFINE and EDIT in Chapter 7 for more information.

Saving Commands for Later Use

Through SQL*Plus, you can store one or more commands in a file called a *command file*. After you create a command file, you can retrieve, edit, and run it. Use command files to save commands for use over time, especially complex commands or PL/SQL blocks.

Storing Commands in Command Files

You can store one or more SQL commands, PL/SQL blocks, and SQL*Plus commands in command files. You create a command file within SQL*Plus in one of three ways:

- enter a command and save the contents of the buffer
- use INPUT to enter commands and then save the buffer contents
- use EDIT to create the file from scratch using a host system text editor

Because SQL*Plus commands are not stored in the buffer, you must use one of the latter two methods to save SQL*Plus commands.

Creating a Command File by Saving the Buffer Contents

To save the current SQL command or PL/SQL block for later use, enter the SAVE command. Follow the command with a file name:

```
SQL> SAVE file_name
```

SQL*Plus adds the extension SQL to the filename to identify it as a SQL query file. If you wish to save the command or block under a name with a different file extension, type a period at the end of the filename, followed by the extension you wish to use.

Note that within SQL*Plus, you separate the extension from the filename with a period. Your operating system may use a different character or a space to separate the filename and the extension.

Example 3–6
Saving the Current Command

First, LIST the buffer contents to see your current command:

```
SQL> LIST
  1   SELECT DEPTNO, ENAME, SAL
  2   FROM EMP
  3   WHERE DEPTNO = 10
  4*  ORDER BY SAL DESC
```

If the query shown is not in your buffer, re-enter the query now. Next, enter the SAVE command followed by the filename DEPTINFO:

```
SQL> SAVE DEPTINFO
Created file DEPTINFO
```

You can verify that the command file DEPTINFO exists by entering the SQL*Plus HOST command followed by your host operating system's file listing command:

```
SQL> HOST your_host's_file_listing_command
```

You can use the same method to save a PL/SQL block currently stored in the buffer.

Creating a Command File by Using INPUT and SAVE

If you use INPUT to enter your commands, you can enter SQL*Plus commands (as well as one or more SQL commands or PL/SQL blocks) into the buffer. You must enter the SQL*Plus commands first, and the SQL command(s) or PL/SQL block(s) last—just as you would if you were entering the commands directly to the command prompt.

You can also store a set of SQL*Plus commands you plan to use with many different queries by themselves in a command file.

Example 3–7
Saving Commands
Using INPUT and
SAVE

Suppose you have composed a query to display a list of salespeople and their commissions. You plan to run it once a month to keep track of how well each employee is doing. To compose and save the query using INPUT, you must first clear the buffer:

```
SQL> CLEAR BUFFER
```

Next, use INPUT to enter the command (be sure not to type a semicolon at the end of the command):

```
SQL> INPUT
  1  COLUMN ENAME HEADING SALESMAN
  2  COLUMN SAL HEADING SALARY FORMAT $99,999
  3  COLUMN COMM HEADING COMMISSION FORMAT $99,990
  4  SELECT EMPNO, ENAME, SAL, COMM
  5  FROM EMP
  6  WHERE JOB = 'SALESMAN'
  7
```

The zero at the end of the format model for the column COMM tells SQL*Plus to display a zero instead of a blank when the value of COMM is zero for a given row. Format models and the COLUMN command are described in more detail in Chapter 4.

Now use the SAVE command to store your query in a file called SALES with the extension SQL:

```
SQL> SAVE SALES
Created file SALES
```

Note that you do not type a semicolon at the end of the query; if you did include a semicolon, SQL*Plus would attempt to run the buffer contents. The SQL*Plus commands in the buffer would produce an error because SQL*Plus expects to find only SQL commands in the buffer. You will learn how to run a command file later in this chapter.

To input more than one SQL command, leave out the semicolons on all the SQL commands. Then, use APPEND to add a semicolon to all but the last command. (SAVE appends a slash to the end of the file automatically; this slash tells SQL*Plus to run the last command when you run the command file.)

To input more than one PL/SQL block, enter the blocks one after another without including a period or a slash on a line between blocks. Then, for each block except the last, list the last line of the block to make it current and use INPUT in the following form to insert a slash on a line by itself:

```
INPUT /
```

Creating Command Files with a System Editor	You can also create a command file with a host operating system text editor by entering EDIT followed by the name of the file, for example:

```
SQL> EDIT SALES
```

Like the SAVE command, EDIT adds the filename extension SQL to the name unless you type a period and a different extension at the end of the filename. When you save the command file with the text editor, it is saved back into the same file.

You must include a semicolon at the end of each SQL command and a period on a line by itself after each PL/SQL block in the file. (You can include multiple SQL commands and PL/SQL blocks.)

When you create a command file using EDIT, you can also include SQL*Plus commands at the end of the file. You cannot do this when you create a command file using the SAVE command because SAVE appends a slash to the end of the file. This slash would cause SQL*Plus to run the command file twice, once upon reaching the semicolon at the end of the last SQL command (or the slash after the last PL/SQL block) and once upon reaching the slash at the end of the file.

Placing Comments in Command Files

You can enter comments in a command file in one of three ways:

- using the SQL*Plus REMARK command
- using the SQL comment delimiters, /* ... */
- using ANSI/ISO (American National Standards Institute/International Standards Organization) comments, – –

Anything that is identified in one of these ways as a comment is not parsed or executed by SQL*Plus.

> **Note:** You cannot enter a comment on the same line after a semicolon.

Using the REMARK Command

Use the REMARK command on a line by itself in the command file, followed by comments on the same line. To continue the comments on additional lines, enter additional REMARK commands. Do not place a REMARK command between different lines of a single SQL command.

```
REMARK Commissions report
REMARK to be run monthly.
COLUMN ENAME HEADING SALESMAN
COLUMN SAL HEADING SALARY FORMAT $99,999
COLUMN COMM HEADING COMMISSION FORMAT $99,990
REMARK Includes only salesmen.
```

```
SELECT EMPNO, ENAME, SAL, COMM
FROM EMP
WHERE JOB = 'SALESMAN'
```

Using /*...*/   Enter the SQL comment delimiters, /*...*/, on separate lines in your command file, on the same line as a SQL command, or on a line in a PL/SQL block. The comments can span multiple lines, but cannot be nested within one another:

```
/* Commissions report
to be run monthly. */
COLUMN ENAME HEADING SALESMAN
COLUMN SAL HEADING SALARY FORMAT $99,999
COLUMN COMM HEADING COMMISSION FORMAT $99,990
SELECT EMPNO, ENAME, SAL, COMM
FROM EMP
WHERE JOB = 'SALESMAN' /* Includes only salesmen. */
```

If you enter a SQL comment directly at the command prompt, SQL*Plus does not store the comment in the buffer.

Using – –   You can use ANSI/ISO "– –" style comments within SQL statements, PL/SQL blocks, or SQL*Plus commands. Since there is no ending delimiter, the comment cannot span multiple lines. For PL/SQL and SQL, enter the comment after a command on a line, or on a line by itself:

```
-- Commissions report to be run monthly
DECLARE  --block for reporting monthly sales
```

For SQL*Plus commands, you can only include "– –" style comments if they are on a line by themselves. For example, these comments are legal:

```
--set maximum width for LONG to 777
SET LONG 777
-- set the heading for ENAME to be SALESMAN
COLUMN ENAME HEADING SALESMAN
```

These comments are illegal:

```
SET LONG 777 -- set maximum width for LONG to 777
SET -- set maximum width for LONG to 777 LONG 777
```

If you entered the following SQL*Plus command, it would be treated as a comment and would not be executed:

```
-- SET LONG 777
```

Retrieving Command Files

If you want to place the contents of a command file in the buffer, you must retrieve the command from the file in which it is stored. You can retrieve a command file using the SQL*Plus command GET.

Just as you can save a query from the buffer to a file with the SAVE command, you can retrieve a query from a file to the buffer with the GET command:

```
SQL> GET file_name
```

When appropriate to the operating system, SQL*Plus adds a period and the extension SQL to the filename unless you type a period at the end of the filename followed by a different extension.

Example 3–8 Retrieving a Command File

Suppose you need to retrieve the SALES file in a later session. You can retrieve the file by entering the GET command. To retrieve the file SALES, enter

```
SQL> GET SALES
  1   COLUMN ENAME HEADING SALESMAN
  2   COLUMN SAL HEADING SALARY FORMAT $99,999
  3   COLUMN COMM HEADING COMMISSION FORMAT $99,990
  4   SELECT EMPNO, ENAME, SAL, COMM
  5   FROM EMP
  6*  WHERE JOB = 'SALESMAN'
```

SQL*Plus retrieves the contents of the file SALES with the extension SQL into the SQL buffer and lists it on the screen. Then you can edit the command further. If the file did not contain SQL*Plus commands, you could also execute it with the RUN command.

Running Command Files

The START command retrieves a command file and runs the command(s) it contains. Use START to run a command file containing SQL commands, PL/SQL blocks, and/or SQL*Plus commands. Follow the word START with the name of the file:

```
START file_name
```

If the file has the extension SQL, you need not add the period and the extension SQL to the filename.

Example 3–9
Running a Command File

To retrieve and run the command stored in SALES.SQL, enter

```
SQL> START SALES
```

SQL*Plus runs the commands in the file SALES and displays the results of the commands on your screen, formatting the query results according to the SQL*Plus commands in the file:

```
     EMPNO SALESMAN     SALARY COMMISSION
---------- ---------- -------- ----------
      7499 ALLEN       $1,600       $300
      7521 WARD        $1,250       $500
      7654 MARTIN      $1,250     $1,400
      7844 TURNER      $1,500         $0
```

To see the commands as SQL*Plus "enters" them, you can set the ECHO variable of the SET command to ON. The ECHO variable controls the listing of the commands in command files run with the START, @ and @@ commands. Setting the ECHO variable to OFF suppresses the listing.

You can also use the @ ("at" sign) command to run a command file:

```
SQL> @SALES
```

The @ command lists and runs the commands in the specified command file in the same manner as START. SET ECHO affects the @ command as it affects the START command.

START, @ and @@ leave the last SQL command or PL/SQL block in the command file in the buffer.

Running a Command File as You Start SQL*Plus

To run a command file as you start SQL*Plus, use one of the following four options:

- Follow the SQLPLUS command with your username, a slash, your password, a space, @, and the name of the file:

```
SQLPLUS SCOTT/TIGER  @SALES
```

  SQL*Plus starts and runs the command file.

- Follow the SQLPLUS command and your username with a space, @, and the name of the file:

```
SQLPLUS SCOTT  @SALES
```

  SQL*Plus prompts you for your password, starts, and runs the command file.

- Include your username as the first line of the file. Follow the SQLPLUS command with @ and the filename. SQL*Plus prompts for your password, starts, and runs the file.

- Include your username, a slash (/), and your password as the first line of the file. Follow the SQLPLUS command with @ and the filename. SQL*Plus starts and runs the file.

Nesting Command Files

To run a series of command files in sequence, first create a command file containing several START commands, each followed by the name of a command file in the sequence. Then run the command file containing the START commands. For example, you could include the following START commands in a command file named SALESRPT:

```
START Q1SALES
START Q2SALES
START Q3SALES
START Q4SALES
START YRENDSLS
```

> **Note:** The @@ command may be useful in this example. See the @@ command in Chapter 7 for more information.

Modifying Command Files

You can modify an existing command file in two ways:

- using the EDIT command
- using GET, the SQL*Plus editing commands, and SAVE

To edit an existing command file with the EDIT command, follow the word EDIT with the name of the file. For example, to edit an existing file named PROFIT that has the extension SQL, enter the following command:

```
SQL> EDIT PROFIT
```

Remember that EDIT assumes the file extension SQL if you do not specify one.

To edit an existing file using GET, the SQL*Plus editing commands, and SAVE, first retrieve the file with GET, then edit the file with the SQL*Plus editing commands, and finally save the file with the SAVE command.

Note that if you want to replace the contents of an existing command file with the command or block in the buffer, you must use the SAVE command and follow the filename with the word REPLACE.

For example:

```
SQL> GET MYREPORT
  1* SELECT * FROM EMP
SQL> C/*/ENAME, JOB
  1* SELECT ENAME, JOB FROM EMP
SQL> SAVE MYREPORT REPLACE
Wrote file MYREPORT
```

If you want to append the contents of the buffer to the end of an existing command file, use the SAVE command and follow the filename with the word APPEND:

```
SQL> SAVE file_name APPEND
```

Exiting from a Command File with a Return Code

If your command file generates a SQL error while running from a batch file on the host operating system, you may want to abort the command file and exit with a return code. Use the SQL*Plus command WHENEVER SQLERROR to do this; see WHENEVER SQLERROR in Chapter 7 for more information.

Similarly, the WHENEVER OSERROR command may be used to exit if an operating system error occurs. See WHENEVER OSERROR in Chapter 7 for more information.

Setting Up Your SQL*Plus Environment

You may wish to set up your SQL*Plus environment in a particular way (such as showing the current time as part of the SQL*Plus command prompt) and then reuse those settings with each session. You can do this through a host operating system file called LOGIN with the file extension SQL (also called your *User Profile*). The exact name of this file is system dependent; see the Oracle installation and user's manual(s) provided for your operating system for the precise name.

You can add any SQL commands, PL/SQL blocks, or SQL*Plus commands to this file; when you start SQL*Plus, it automatically searches for your LOGIN file (first in your local directory and then on a system-dependent path) and runs the commands it finds there. (You may also have a Site Profile, for example, GLOGIN.SQL. See the SQLPLUS command in Chapter 6 for more information on the relationship of Site and User Profiles.)

Modifying Your LOGIN File	You can modify your LOGIN file just as you would any other command file. You may wish to add some of the following commands to the LOGIN file:

`SET COMPATIBILITY`	Followed by V7or V8, sets compatibility to the version of Oracle you specify. Setting COMPATIBILITY to V7 allows you to run command files created with Oracle7.
`SET NUMFORMAT`	Followed by a number format (such as $99,999), sets the default format for displaying numbers in query results.
`SET PAGESIZE`	Followed by a number, sets the number of lines per page.
`SET PAUSE`	Followed by ON, causes SQL*Plus to pause at the beginning of each page of output (SQL*Plus continues scrolling after you enter [Return]). Followed by text, sets the text to be displayed each time SQL*Plus pauses (you must also set PAUSE to ON).
`SET SHIFTINOUT`	Followed by VISIBLE, will display shift characters as a visible character. Setting SHIFTINOUT to INVISIBLE, will not display any shift characters. Note, this command can only be used with shift sensitive character sets.
`SET TIME`	Followed by ON, displays the current time before each command prompt.

See the SET command in Chapter 7 for more information on these and other SET command variables you may wish to set in your SQL*Plus LOGIN file.

Storing and Restoring SQL*Plus System Variables

You can store the current SQL*Plus system ("SET") variables in a host operating system file (a command file) with the STORE command. If you alter any variables, this command file can be run to restore the original values. This is useful if you run a report that alters system variables and you want to reset their values after the report has finished.

To store the current setting of all system variables, enter

```
SQL> STORE SET file_name
```

By default, SQL*Plus adds the extension "SQL" to the file name. If you want to use a different file extension, type a period at the end of the file name, followed by the extension. Alternatively, you can use the SET SUFFIX command to change the default file extension.

Restoring the System Variables	To restore the stored system variables, enter

```
SQL> START file_name
```

If the file has the default extension (as specified by the SET SUFFIX command), you do not need to add the period and extension to the file name.

You can also use the @ ("at" sign) or the @@ (double "at" sign) commands to run the command file.

Example 3–10 Storing and Restoring SQL*Plus System Variables	To store the current values of the SQL*Plus system variables in a new command file "plusenv.sql":

```
SQL> STORE SET plusenv
Created file plusenv
```

Now the value of any system variable can be changed:

```
SQL> SHOW PAGESIZE
pagesize 24
SQL> SET PAGESIZE 60
SQL> SHOW PAGESIZE
pagesize 60
```

The original values of the system variables can then be restored from the command file:

```
SQL> START plusenv
SQL> SHOW PAGESIZE
pagesize 24
```

Writing Interactive Commands

The following features of SQL*Plus make it possible for you to set up command files that allow end-user input:

- defining user variables
- substituting values in commands
- using the START command to provide values
- prompting for values

Defining User Variables	You can define variables, called *user variables*, for repeated use in a single command file by using the SQL*Plus command DEFINE. Note that you can also define user variables to use in titles and to save you keystrokes (by defining a long string as the value for a variable with a short name).

**Example 3–11
Defining a User
Variable**

To define a user variable EMPLOYEE and give it the value "SMITH", enter the following command:

```
SQL> DEFINE EMPLOYEE = SMITH
```

To confirm the definition of the variable, enter DEFINE followed by the variable name:

```
SQL> DEFINE EMPLOYEE
```

SQL*Plus lists the definition:

```
DEFINE EMPLOYEE        = "SMITH" (CHAR)
```

To list all user variable definitions, enter DEFINE by itself at the command prompt. Note that any user variable you define explicitly through DEFINE takes only CHAR values (that is, the value you assign to the variable is always treated as a CHAR datatype). You can define a user variable of datatype NUMBER implicitly through the ACCEPT command. You will learn more about the ACCEPT command later in this chapter.

To delete a user variable, use the SQL*Plus command UNDEFINE followed by the variable name.

Using Substitution Variables

Suppose you want to write a query like the one in SALES (see Example 3–7) to list the employees with various jobs, not just those whose job is SALESMAN. You could do that by editing a different CHAR value into the WHERE clause each time you run the command, but there is an easier way.

By using a *substitution variable* in place of the value SALESMAN in the WHERE clause, you can get the same results you would get if you had written the values into the command itself.

A substitution variable is a user variable name preceded by one or two ampersands (&). When SQL*Plus encounters a substitution variable in a command, SQL*Plus executes the command as though it contained the value of the substitution variable, rather than the variable itself.

For example, if the variable SORTCOL has the value JOB and the variable MYTABLE has the value EMP, SQL*Plus executes the commands

```
SQL> BREAK ON &SORTCOL
SQL> SELECT &SORTCOL, SAL
  2  FROM &MYTABLE
  3  ORDER BY &SORTCOL;
```

as if they were

```
SQL> BREAK ON JOB
SQL> SELECT JOB, SAL
  2  FROM EMP
  3  ORDER BY JOB;
```

(The BREAK command suppresses duplicate values of the column named in SORTCOL. For more information about the BREAK command, see the section "Clarifying Your Report with Spacing and Summary Lines" in Chapter 4.)

Where and How to Use Substitution Variables

You can use substitution variables anywhere in SQL and SQL*Plus commands, except as the first word entered at the command prompt. When SQL*Plus encounters an undefined substitution variable in a command, SQL*Plus prompts you for the value.

You can enter any string at the prompt, even one containing blanks and punctuation. If the SQL command containing the reference should have quote marks around the variable and you do not include them there, the user must include the quotes when prompted.

SQL*Plus reads your response from the keyboard, even if you have redirected terminal input or output to a file. If a terminal is not available (if, for example, you run the command file in batch mode), SQL*Plus uses the redirected file.

After you enter a value at the prompt, SQL*Plus lists the line containing the substitution variable twice: once before substituting the value you enter and once after substitution. You can suppress this listing by setting the SET command variable VERIFY to OFF.

**Example 3–12
Using Substitution
Variables**

Create a command file named STATS, to be used to calculate a subgroup statistic (the maximum value) on a numeric column:

```
SQL> CLEAR BUFFER
SQL> INPUT
  1  SELECT   &GROUP_COL,
  2           MAX(&NUMBER_COL) MAXIMUM
  3  FROM     &TABLE
  4  GROUP BY &GROUP_COL
  5
SQL> SAVE STATS
Created file STATS
```

Now run the command file STATS and respond as shown below to the prompts for values:

```
SQL>  @STATS
Enter value for group_col: JOB
old   1: SELECT   &GROUP_COL,
new   1: SELECT   JOB,
Enter value for number_col: SAL
old   2:          MAX(&NUMBER_COL) MAXIMUM
new   2:          MAX(SAL) MAXIMUM
Enter value for table: EMP
old   3: FROM     &TABLE
new   3: FROM     EMP
Enter value for group_col: JOB
old   4: GROUP BY &GROUP_COL
new   4: GROUP BY JOB
```

SQL*Plus displays the following output:

```
JOB          MAXIMUM
---------- ----------
ANALYST         3000
CLERK           1300
MANAGER         2975
PRESIDENT       5000
SALESMAN        1600
```

If you wish to append characters immediately after a substitution variable, use a period to separate the variable from the character. For example:

```
SQL> SELECT * FROM EMP WHERE EMPNO='&X.01';
Enter value for X:   123
```

will be interpreted as

```
SQL> SELECT * FROM EMP WHERE EMPNO='12301';
```

Avoiding Unnecessary Prompts for Values

Suppose you wanted to expand the file STATS to include the minimum, sum, and average of the "number" column. You may have noticed that SQL*Plus prompted you twice for the value of GROUP_COL and once for the value of NUMBER_COL in Example 3–12, and that each GROUP_COL or NUMBER_COL had a single ampersand in front of it. If you were to add three more functions—using a single ampersand before each—to the command file, SQL*Plus would prompt you a total of four times for the value of the number column.

You can avoid being re-prompted for the group and number columns by adding a second ampersand in front of each GROUP_COL and NUMBER_COL in STATS. SQL*Plus automatically DEFINEs any substitution variable preceded by two ampersands, but does not DEFINE those preceded by only one ampersand. When you have DEFINEd a variable, SQL*Plus substitutes the value of *variable* for each substitution variable referencing *variable* (in the form &*variable* or &&*variable*). SQL*Plus will not prompt you for the value of *variable* in this session until you UNDEFINE *variable*.

Example 3–13
Using Double
Ampersands

To expand the command file STATS using double ampersands and then run the file, first suppress the display of each line before and after substitution:

```
SQL> SET VERIFY OFF
```

Now retrieve and edit STATS by entering the following commands:

```
SQL> GET STATS
  1  SELECT &GROUP_COL,
  2         MAX(&NUMBER_COL) MAXIMUM
  3  FROM   &TABLE
  4  GROUP BY &GROUP_COL
SQL> 2
  2*  MAX(&NUMBER_COL) MAXIMUM
SQL> APPEND ,
  2*  MAX(&NUMBER_COL) MAXIMUM,
SQL> C /&/&&
  2*  MAX(&&NUMBER_COL) MAXIMUM,
SQL> I
  3i  MIN(&&NUMBER_COL) MINIMUM,
  4i  SUM(&&NUMBER_COL) TOTAL,
  5i  AVG(&&NUMBER_COL) AVERAGE
  6i
SQL> 1
  1* SELECT &GROUP_COL,
SQL> C /&/&&
  1* SELECT &&GROUP_COL,
SQL> 7
  7*  GROUP BY &GROUP_COL
SQL> C /&/&&
  7*  GROUP BY &&GROUP_COL
SQL> SAVE STATS2
created file STATS2
```

Finally, run the command file STATS2 and respond to the prompts for values as follows:

```
SQL> START STATS2
Enter value for group_col: JOB
Enter value for number_col: SAL
Enter value for table: EMP
```

SQL*Plus displays the following output:

```
JOB           MAXIMUM    MINIMUM      TOTAL    AVERAGE
---------- ---------- ---------- ---------- ---------
ANALYST          3000       3000       6000      3000
CLERK            1300        800       4150    1037.5
MANAGER          2975       2450       8275 2758.33333
PRESIDENT        5000       5000       5000      5000
SALESMAN         1600       1250       5600      1400
```

Note that you were prompted for the values of NUMBER_COL and GROUP_COL only once. If you were to run STATS2 again during the current session, you would be prompted for TABLE (because its name has a single ampersand and the variable is therefore not DEFINEd) but not for GROUP_COL or NUMBER_COL (because their names have double ampersands and the variables are therefore DEFINEd).

Before continuing, set the system variable VERIFY back to ON:

```
SQL> SET VERIFY ON
```

Restrictions

You cannot use substitution variables in the buffer editing commands, APPEND, CHANGE, DEL, and INPUT, nor in other commands where substitution would be meaningless, such as REMARK. The buffer editing commands, APPEND, CHANGE, and INPUT, treat text beginning with "&" or "&&" literally, as any other text string.

System Variables

The following system variables, specified with the SQL*Plus SET command, affect substitution variables:

SET DEFINE   Defines the substitution character (by default the ampersand "&") and turns substitution on and off.

SET ESCAPE   Defines an escape character you can use before the substitution character. The escape character instructs SQL*Plus to treat the substitution character as an ordinary character rather than as a request for variable substitution. The default escape character is a backslash (\).

SET VERIFY ON	Lists each line of the command file before and after substitution.
SET CONCAT	Defines the character that separates the name of a substitution variable or parameter from characters that immediately follow the variable or parameter—by default the period (.).

Refer to SET in Chapter 7 for more information on these system variables.

Passing Parameters through the START Command

You can bypass the prompts for values associated with substitution variables by passing values to parameters in a command file through the START command.

You do this by placing an ampersand (&) followed by a numeral in the command file in place of a substitution variable. Each time you run this command file, START replaces each &1 in the file with the first value (called an argument) after START *filename*, then replaces each &2 with the second value, and so forth.

For example, you could include the following commands in a command file called MYFILE:

```
SELECT * FROM EMP
WHERE JOB='&1'
AND SAL=&2
```

In the following START command, SQL*Plus would substitute CLERK for &1 and 7900 for &2 in the command file MYFILE:

```
SQL> START MYFILE CLERK 7900
```

When you use arguments with the START command, SQL*Plus DEFINEs each parameter in the command file with the value of the appropriate argument.

Example 3–14 Passing Parameters through START

To create a new command file based on SALES that takes a parameter specifying the job to be displayed, enter

```
SQL> GET SALES
  1   COLUMN ENAME HEADING SALESMAN
  2   COLUMN SAL HEADING SALARY FORMAT $99,999
  3   COLUMN COMM HEADING COMMISSION FORMAT $99,990
  4   SELECT EMPNO, ENAME, SAL, COMM
  5   FROM EMP
  6* WHERE JOB = 'SALESMAN'
SQL> CHANGE /SALESMAN/&1
  6* WHERE  JOB = '&1'
```

```
SQL> 1
  1* COLUMN ENAME HEADING SALESMAN
SQL> CHANGE /SALESMAN/&1
  1* COLUMN ENAME HEADING &1
SQL> SAVE ONEJOB
Created file ONEJOB
```

Now run the command with the parameter CLERK:

```
SQL> START ONEJOB CLERK
```

SQL*Plus lists the line of the SQL command that contains the parameter, before and after replacing the parameter with its value, and then displays the output:

```
old    3: WHERE JOB = '&1'
new    3: WHERE JOB = 'CLERK'
    EMPNO CLERK          SALARY COMMISSION
--------- ---------- ---------- ----------
     7369 SMITH             $800
     7876 ADAMS           $1,100
     7900 JAMES             $950
     7934 MILLER          $1,300
```

You can use any number of parameters in a command file. Within a command file, you can refer to each parameter any number of times, and can include the parameters in any order.

> **Note:** You cannot use parameters when you run a command with RUN or slash (/). You must store the command in a command file and run it with START or @.

Before continuing, return the column ENAME to its original heading by entering the following command:

```
SQL> COLUMN ENAME CLEAR
```

Communicating with the User

Three SQL*Plus commands—PROMPT, ACCEPT, and PAUSE—help you communicate with the end user. These commands enable you to send messages to the screen and receive input from the user, including a simple [Return]. You can also use PROMPT and ACCEPT to customize the prompts for values SQL*Plus automatically generates for substitution variables.

Prompting for and Accepting User Variable Values

Through PROMPT and ACCEPT, you can send messages to the end user and accept values as end-user input. PROMPT simply displays a message you specify on-screen; use it to give directions or information to the user. ACCEPT prompts the user for a value and stores it in the user variable you specify. Use PROMPT in conjunction with ACCEPT when your prompt for the value spans more than one line.

Example 3–15
Prompting for and
Accepting Input

To direct the user to supply a report title and to store the input in the variable MYTITLE for use in a subsequent query, first clear the buffer:

```
SQL> CLEAR BUFFER
```

Next, set up a command file as shown below:

```
SQL> INPUT
  1  PROMPT Enter a title up to 30 characters long.
  2  ACCEPT MYTITLE PROMPT 'Title:  '
  3  TTITLE LEFT MYTITLE SKIP 2
  4  SELECT * FROM DEPT
  5
SQL> SAVE PROMPT1
Created file PROMPT1
```

The TTITLE command sets the top title for your report. This command is covered in detail in Chapter 4.

Finally, run the command file, responding to the prompt for the title as shown:

```
SQL> START PROMPT1
Enter a title up to 30 characters long.
Title:  Department Report as of 1/1/95
```

SQL*Plus displays the following output:

```
Department Report as of 1/1/95
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Before continuing, turn the TTITLE command you entered in the command file off as shown below:

```
SQL> TTITLE OFF
```

Customizing Prompts for Substitution Variable Values	If you want to customize the prompt for a substitution variable value, use PROMPT and ACCEPT in conjunction with the substitution variable, as shown in the following example.

Example 3–16 Using PROMPT and ACCEPT in Conjunction with Substitution Variables

As you have seen in Example 3–15, SQL*Plus automatically generates a prompt for a value when you use a substitution variable. You can replace this prompt by including PROMPT and ACCEPT in the command file with the query that references the substitution variable. To create such a file, enter the commands shown:

```
SQL> CLEAR BUFFER
buffer cleared
SQL> INPUT
  1  PROMPT Enter a valid employee number
  2  PROMPT For example:  7123, 7456, 7890
  3  ACCEPT ENUMBER NUMBER PROMPT 'Emp. no.:  '
  4  SELECT ENAME, MGR, JOB, SAL
  5  FROM EMP
  6  WHERE EMPNO = &ENUMBER
  7
SQL> SAVE PROMPT2
Created file PROMPT2
```

Next, run the command file. SQL*Plus prompts for the value of ENUMBER using the text you specified with PROMPT and ACCEPT:

```
SQL> START PROMPT2
Enter a valid employee number
For example:  7123, 7456, 7890
Emp. No.:
```

Try entering characters instead of numbers to the prompt for "Emp. No.":

```
Emp. No.:  ONE
"ONE" is not a valid number
Emp. No.:
```

Because you specified NUMBER after the variable name in the ACCEPT command, SQL*Plus will not accept a non-numeric value. Now enter a number:

```
Emp. No.:  7521
old   3: WHERE EMPNO = &ENUMBER
new   3: WHERE EMPNO =        7521
```

SQL*Plus displays the following output:

```
ENAME            MGR JOB         SALARY
---------- ---------- --------- ----------
WARD            7698 SALESMAN     $1,250
```

Sending a Message and Accepting [Return] as Input

If you want to display a message on the user's screen and then have the user enter [Return] after reading the message, use the SQL*Plus command PAUSE. For example, you might include the following lines in a command file:

```
PROMPT Before continuing, make sure you have your account
card.
PAUSE Press RETURN to continue.
```

Clearing the Screen

If you want to clear the screen before displaying a report (or at any other time), include the SQL*Plus CLEAR command with its SCREEN clause at the appropriate point in your command file, using the following format:

```
CLEAR SCREEN
```

Before continuing to the next chapter, reset all columns to their original formats and headings by entering the following command:

```
SQL> CLEAR COLUMNS
```

Using Bind Variables

Suppose that you want to be able to display the variables you use in your PL/SQL subprograms in SQL*Plus or use the same variables in multiple subprograms. If you declare a variable in a PL/SQL subprogram, you cannot display that variable in SQL*Plus. Use a bind variable in PL/SQL to access the variable from SQL*Plus.

Bind variables are variables you create in SQL*Plus and then reference in PL/SQL. If you create a bind variable in SQL*Plus, you can use the variable as you would a declared variable in your PL/SQL subprogram and then access the variable from SQL*Plus. You can use bind variables for such things as storing return codes or debugging your PL/SQL subprograms.

Because bind variables are recognized by SQL*Plus, you can display their values in SQL*Plus or reference them in other PL/SQL subprograms that you run in SQL*Plus.

Creating Bind Variables

You create bind variables in SQL*Plus with the VARIABLE command. For example

```
VARIABLE ret_val NUMBER
```

This command creates a bind variable named ret_val with a datatype of NUMBER. See VARIABLE in Chapter 7. (To list all of the bind variables created in a session, type VARIABLE without any arguments.)

Referencing Bind Variables

You reference bind variables in PL/SQL by typing a colon (:) followed immediately by the name of the variable. For example

```
:ret_val := 1;
```

This command assigns a value to the bind variable named ret_val.

Displaying Bind Variables

To display the value of a bind variable in SQL*Plus, you use the SQL*Plus PRINT command. For example

```
PRINT ret_val
```

This command displays a bind variable named ret_val. See PRINT in Chapter 7.

Example 3–17 Creating, Referencing, and Displaying Bind Variables

To declare a local bind variable named id with a datatype of NUMBER, enter

```
VARIABLE id NUMBER
```

Next, put a value of "1" into the bind variable you have just created:

```
BEGIN
:id := 1;
END;
```

If you want to display a list of values for the bind variable named id, enter

```
PRINT id
```

Try creating some new departments using the variable:

```
EXECUTE :id := dept_management.new('ACCOUNTING','NEW YORK')
EXECUTE :id := dept_management.new('RESEARCH','DALLAS')
EXECUTE :id := dept_management.new('SALES','CHICAGO')
EXECUTE :id := dept_management.new('OPERATIONS','BOSTON')
PRINT id
COMMIT
```

> **Note:** *dept_management.new* refers to a PL/SQL function, "new", in a package (*dept_management*). The function "new" adds the department data to a table.

Using REFCURSOR Bind Variables

SQL*Plus REFCURSOR bind variables allow SQL*Plus to fetch and format the results of a SELECT statement contained in a PL/SQL block.

REFCURSOR bind variables can also be used to reference PL/SQL cursor variables in stored procedures. This allows you to store SELECT statements in the database and reference them from SQL*Plus.

A REFCURSOR bind variable can also be returned from a stored function.

> **Note:** You must have Oracle7, Release 7.3 or above to assign the return value of a stored function to a REFCURSOR variable.

Example 3–18 Creating, Referencing, and Displaying REFCURSOR Bind Variables

To create, reference and display a REFCURSOR bind variable, first declare a local bind variable of the REFCURSOR datatype

```
SQL> VARIABLE dept_sel REFCURSOR
```

Next, enter a PL/SQL block that uses the bind variable in an OPEN ... FOR SELECT statement. This statement opens a cursor variable and executes a query. See the *PL/SQL User's Guide and Reference* for information on the OPEN command and cursor variables.

In this example we are binding the SQL*Plus *dept_sel* bind variable to the cursor variable.

```
SQL> BEGIN
  2    OPEN :dept_sel FOR SELECT * FROM DEPT;
  3  END;
  4  /
PL/SQL procedure successfully completed.
```

The results from the SELECT statement can now be displayed in SQL*Plus with the PRINT command.

```
SQL> PRINT dept_sel
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

The PRINT statement also closes the cursor. To reprint the results, the PL/SQL block must be executed again before using PRINT.

A REFCURSOR bind variable is passed as a parameter to a procedure.
The parameter has a REF CURSOR type. First, define the type.

```
SQL> CREATE OR REPLACE PACKAGE cv_types AS
  2    TYPE DeptCurTyp is REF CURSOR RETURN dept%ROWTYPE;
  3  END cv_types;
  4  /
Package created.
```

Next, create the stored procedure containing an OPEN ... FOR SELECT
statement.

```
SQL> CREATE OR REPLACE PROCEDURE dept_rpt
  2    (dept_cv IN OUT cv_types.DeptCurTyp) AS
  3  BEGIN
  4    OPEN dept_cv FOR SELECT * FROM DEPT;
  5  END;
  6  /
Procedure successfully completed.
```

Execute the procedure with a SQL*Plus bind variable as the parameter.

```
SQL> VARIABLE odcv REFCURSOR
SQL> EXECUTE dept_rpt(:odcv)
PL/SQL procedure successfully completed.
```

Now print the bind variable.

```
SQL> PRINT odcv
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

The procedure can be executed multiple times using the same or a
different REFCURSOR bind variable.

```
SQL> VARIABLE pcv REFCURSOR
SQL> EXECUTE dept_rpt(:pcv)
PL/SQL procedure successfully completed.
SQL> PRINT pcv
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Example 3–20
Using REFCURSOR
Variables in Stored
Functions
Create a stored function containing an OPEN ... FOR SELECT statement:

```
SQL> CREATE OR REPLACE FUNCTION dept_fn RETURN –
>      cv_types.DeptCurTyp IS
  2    resultset cv_types.DeptCurTyp;
  3  BEGIN
  4    OPEN resultset FOR SELECT * FROM DEPT;
  5    RETURN(resultset);
  6  END;
  7  /
Function created.
```

Execute the function.

```
SQL> VARIABLE rc REFCURSOR
SQL> EXECUTE :rc := dept_fn
PL/SQL procedure successfully completed.
```

Now print the bind variable.

```
SQL> PRINT rc
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
4 rows selected.
```

The function can be executed multiple times using the same or a different REFCURSOR bind variable.

```
SQL> EXECUTE :rc := dept_fn
PL/SQL procedure successfully completed.
SQL> PRINT rc
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
4 rows selected.
```

Tracing Statements

You can automatically get a report on the execution path used by the SQL optimizer and the statement execution statistics. The report is generated after successful SQL DML (that is, SELECT, DELETE, UPDATE and INSERT) statements. It is useful for monitoring and tuning the performance of these statements.

Controlling the Report You can control the report by setting the AUTOTRACE system variable.

`SET AUTOTRACE OFF`	No AUTOTRACE report is generated. This is the default.
`SET AUTOTRACE ON EXPLAIN`	The AUTOTRACE report shows only the optimizer execution path.
`SET AUTOTRACE ON STATISTICS`	The AUTOTRACE report shows only the SQL statement execution statistics.
`SET AUTOTRACE ON`	The AUTOTRACE report includes both the optimizer execution path and the SQL statement execution statistics.
`SET AUTOTRACE TRACEONLY`	Like SET AUTOTRACE ON, but suppresses the printing of the user's query output, if any.

To use this feature, you must have the PLUSTRACE role granted to you and a PLAN_TABLE table created in your schema. For more information on the PLUSTRACE role and PLAN_TABLE table, see the AUTOTRACE variable of the SET command in Chapter 7.

Execution Plan The Execution Plan shows the SQL optimizer's query execution path. Both tables are accessed by a full table scan, sorted, and then merged.

Each line of the Execution Plan has a sequential line number. SQL*Plus also displays the line number of the parent operation.

The Execution Plan consists of four columns displayed in the following order:

Column Name	Description
ID_PLUS_EXP	Shows the line number of each execution step.
PARENT_ID_PLUS_EXP	Shows the relationship between each step and its parent. This column is useful for large reports.
PLAN_PLUS_EXP	Shows each step of the report.
OBJECT_NODE_PLUS_EXP	Shows the database links or parallel query servers used.

The format of the columns may be altered with the COLUMN command. For example, to stop the PARENT_ID_PLUS_EXP column being displayed, enter

```
SQL> COLUMN PARENT_ID_PLUS_EXP NOPRINT
```

The default formats can be found in the site profile (for example, glogin.sql).

The Execution Plan output is generated using the EXPLAIN PLAN command. For information about interpreting the output of EXPLAIN PLAN, see the *Oracle8 Server Tuning* guide.

Statistics   The statistics are recorded by the server when your statement executes and indicate the system resources required to execute your statement.

The *client* referred to in the statistics is SQL*Plus. *SQL*Net* refers to the generic process communication between SQL*Plus and the server, regardless of whether SQL*Net is installed.

You cannot change the default format of the statistics report.

For more information about the statistics and how to interpret them, see the *Oracle8 Server Tuning* guide.

**Example 3–21
Tracing Statements for
Performance Statistics
and Query Execution
Path**

If the SQL buffer contains the following statement:

```
SQL> SELECT D.DNAME, E.ENAME, E.SAL, E.JOB
  2  FROM EMP E, DEPT D
  3  WHERE E.DEPTNO = D.DEPTNO
```

The statement can be automatically traced when it is run:

```
SQL> SET AUTOTRACE ON
SQL> /
```

```
DNAME          ENAME           SAL JOB
-------------- ---------- ---------- ---------
ACCOUNTING     CLARK          2450 MANAGER
ACCOUNTING     KING           5000 PRESIDENT
ACCOUNTING     MILLER         1300 CLERK
RESEARCH       SMITH           800 CLERK
RESEARCH       ADAMS          1100 CLERK
RESEARCH       FORD           3000 ANALYST
RESEARCH       SCOTT          3000 ANALYST
RESEARCH       JONES          2975 MANAGER
SALES          ALLEN          1600 SALESMAN
SALES          BLAKE          2850 MANAGER
SALES          MARTIN         1250 SALESMAN
SALES          JAMES           950 CLERK
SALES          TURNER         1500 SALESMAN
SALES          WARD           1250 SALESMAN

14 rows selected.

Execution Plan
-----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0  MERGE JOIN
   2    1   SORT (JOIN)
   3    2    TABLE ACCESS (FULL) OF 'DEPT'
   4    1   SORT (JOIN)
   5    4    TABLE ACCESS (FULL) OF 'EMP'
```

```
Statistics
-----------------------------------------------------------
        148  recursive calls
          4  db block gets
         24  consistent gets
          6  physical reads
         43  redo size
        591  bytes sent via SQL*Net to client
        256  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          2  sort (memory)
          0  sort (disk)
         14  rows processed
```

Note: Your output may vary depending on the version of the server to which you are connected and the configuration of the server.

To trace the same statement without displaying the query data:

```
SQL> SET AUTOTRACE TRACEONLY
SQL> /

Execution Plan
-----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1   0   MERGE JOIN
   2   1    SORT (JOIN)
   3   2     TABLE ACCESS (FULL) OF 'DEPT'
   4   1    SORT (JOIN)
   5   4     TABLE ACCESS (FULL) OF 'EMP'

Statistics
-----------------------------------------------------------
          0  recursive calls
          4  db block gets
          2  consistent gets
          0  physical reads
          0  redo size
        599  bytes sent via SQL*Net to client
        256  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          2  sort (memory)
          0  sort (disk)
         14  rows processed
```

This option is useful when you are tuning a large query, but do not want to see the query report.

Example 3–23
Tracing Statements
Using a Database Link

To trace a statement using a database link:

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
SQL> SELECT * FROM EMP@MY_LINK;


Execution Plan
-----------------------------------------------------------
   0      SELECT STATEMENT (REMOTE) Optimizer=CHOOSE
   1    0    TABLE ACCESS (FULL) OF 'EMP'  MY_LINK.DB_DOMAIN
```

The Execution Plan shows the table being accessed on line 1 is via the database link MY_LINK.DB_DOMAIN.

Tracing Parallel and Distributed Queries

When you trace a statement in a parallel or distributed query, the Execution Plan shows the cost based optimizer estimates of the number of rows (the *cardinality*). In general, the cost, cardinality and bytes at each node represent cumulative results. For example, the cost of a join node accounts for not only the cost of completing the join operations, but also the entire costs of accessing the relations in that join.

Lines marked with an asterisk (*) denote a parallel or remote operation. Each operation is explained in the second part of the report. See the *Oracle8 Server Tuning* guide for more information on parallel and distributed operations.

The second section of this report consists of three columns displayed in the following order:

Column Name	Description
ID_PLUS_EXP	Shows the line number of each execution step.
OTHER_TAG_PLUS_EXP	Describes the function of the SQL statement in the OTHER_PLUS_EXP column.
OTHER_PLUS_EXP	Shows the text of the query for the parallel server or remote database.

The format of the columns may be altered with the COLUMN command. The default formats can be found in the site profile (for example, glogin.sql).

> **Note:** You must have Oracle7, Release 7.3 or greater to view the second section of this report.

**Example 3–24
Tracing Statements
With Parallel Query
Option**

To trace a parallel query running the parallel query option:

```
SQL> CREATE TABLE T2_T1 (UNIQUE1 NUMBER) PARALLEL –
>    (DEGREE 6);

Table created.

SQL> CREATE TABLE T2_T2 (UNIQUE1 NUMBER) PARALLEL –
>    (DEGREE 6);

Table created.

SQL> CREATE UNIQUE INDEX D2_I_UNIQUE1 ON D2_T1(UNIQUE1);

Index created.

SQL> SET LONG 500 LONGCHUNKSIZE 500
SQL> SET AUTOTRACE ON EXPLAIN
SQL> SELECT /*+ INDEX(B,D2_I_UNIQUE1) USE_NL(B) ORDERED –
>    */ COUNT (A.UNIQUE1)
  2  FROM D2_T2 A, D2_T1 B
  3  WHERE A.UNIQUE1 = B.UNIQUE1;
```

SQL*Plus displays the following output:

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1
                           Card=263 Bytes=5786)
   1   0   SORT (AGGREGATE)
   2   1    NESTED LOOPS* (Cost=1 Card=263 Bytes=5785)
                                               :Q8200
   3   2      TABLE ACCESS* (FULL) OF 'D2_T2'       :Q8200
   4   2      INDEX* (UNIQUE SCAN) OF 'D2_I_UNIQUE1'
                      (UNIQUE)                       :Q8200
   2 PARALLEL_TO_SERIAL  SELECT /*+ ORDERED NO_EXPAND
                          USE_NL(A2) INDEX(A2) PIV_SSF */
                          COUNT(A1.C0) FROM (SELECT/*+
                          ROWID(A3) */ A3."UNIQUE1" FROM
                          "D2_T2" A3 WHERE ROWID BETWEEN :1
                          AND :2) A1, "D2_T1" A2 WHERE
                          A1.C0=A2."UNIQUE1"
   3 PARALLEL_COMBINED_WITH_PARENT
   4 PARALLEL_COMBINED_WITH_PARENT
```

Line 0 of the Execution Plan shows the cost based optimizer estimates the number of rows at 263, taking 5786 bytes. The total cost of the statement is 1.

Lines 2, 3 and 4 are marked with asterisks, denoting parallel operations. For example, the NESTED LOOPS step on line 2 is a PARALLEL_TO_SERIAL operation. PARALLEL_TO_SERIAL operations execute a SQL statement to produce output serially. Line 2 also shows that the parallel query server had the identifier *Q8200.*

4

Formatting Query Results

This chapter explains how to format your query results to produce a finished report. This chapter covers the following topics:

- formatting columns
- clarifying your report with spacing and summary lines
- defining page and report titles and dimensions
- storing and printing query results

Read this chapter while sitting at your computer and try out the examples shown. Before beginning, make sure you have access to the sample tables described in Chapter 1.

Formatting Columns

Through the SQL*Plus COLUMN command, you can change the column headings and reformat the column data in your query results.

Changing Column Headings

When displaying column headings, you can either use the default heading or you can change it using the COLUMN command. The sections below describe how the default headings are derived and how you can alter them with the COLUMN command.

Default Headings

SQL*Plus uses column or expression names as default column headings when displaying query results. Column names are often short and cryptic, however, and expressions can be hard to understand.

Changing Default Headings

You can define a more useful column heading with the HEADING clause of the COLUMN command, in the format shown below:

```
COLUMN column_name HEADING column_heading
```

See the COLUMN command in Chapter 7 for more details.

Example 4–1
Changing a Column
Heading

To produce a report from EMP with new headings specified for DEPTNO, ENAME, and SAL, enter the following commands:

```
SQL> COLUMN DEPTNO HEADING Department
SQL> COLUMN ENAME HEADING Employee
SQL> COLUMN SAL HEADING Salary
SQL> COLUMN COMM HEADING Commission
SQL> SELECT DEPTNO, ENAME, SAL, COMM
  2   FROM EMP
  3   WHERE JOB = 'SALESMAN';
```

SQL*Plus displays the following output:

```
Department Employee      Salary Commission
---------- ---------- ---------- ----------
        30 ALLEN            1600        300
        30 WARD             1250        500
        30 MARTIN           1250       1400
        30 TURNER           1500          0
```

> **Note:** The new headings will remain in effect until you enter different headings, reset each column's format, or exit from SQL*Plus.

To change a column heading to two or more words, enclose the new heading in single or double quotation marks when you enter the COLUMN command. To display a column heading on more than one line, use a vertical bar (|) where you want to begin a new line. (You can

use a character other than a vertical bar by changing the setting of the HEADSEP variable of the SET command. See the SET command in Chapter 7 for more information.)

To give the column ENAME the heading EMPLOYEE NAME and to split the new heading onto two lines, enter

```
SQL> COLUMN ENAME HEADING 'Employee|Name'
```

Now re-run the query with the slash (/) command:

```
SQL> /
```

SQL*Plus displays the following output:

```
           Employee
Department Name          Salary Commission
---------- ---------- ---------- ----------
        30 ALLEN            1600        300
        30 WARD             1250        500
        30 MARTIN           1250       1400
        30 TURNER           1500          0
```

To change the character used to underline each column heading, set the UNDERLINE variable of the SET command to the desired character.

To change the character used to underline headings to an equal sign and re-run the query, enter the following commands:

```
SQL> SET UNDERLINE =
SQL> /
```

SQL*Plus displays the following results:

```
           Employee
Department Name          Salary Commission
========== ========== ========== ==========
        30 ALLEN            1600        300
        30 WARD             1250        500
        30 MARTIN           1250       1400
        30 TURNER           1500          0
```

Now change the underline character back to a dash:

```
SQL> SET UNDERLINE '-'
```

> **Note:** You must enclose the dash in quotation marks; otherwise, SQL*Plus interprets the dash as a hyphen indicating you wish to continue the command on another line.

Formatting NUMBER Columns

When displaying NUMBER columns, you can either accept the SQL*Plus default display width or you can change it using the COLUMN command. The sections below describe the default display and how you can alter the default with the COLUMN command.

Default Display

A NUMBER column's width equals the width of the heading or the width of the FORMAT plus one space for the sign, whichever is greater. If you do not explicitly use FORMAT, then the column's width will always be at least the value of SET NUMWIDTH.

SQL*Plus normally displays numbers with as many digits as are required for accuracy, up to a standard display width determined by the value of the NUMWIDTH variable of the SET command (normally 10). If a number is larger than the value of SET NUMWIDTH, SQL*Plus rounds the number up or down to the maximum number of characters allowed.

You can choose a different format for any NUMBER column by using a format model in a COLUMN command. A format model is a representation of the way you want the numbers in the column to appear, using 9's to represent digits.

Changing the Default Display

The COLUMN command identifies the column you want to format and the model you want to use, as shown below:

```
COLUMN column_name FORMAT model
```

Use format models to add commas, dollar signs, angle brackets (around negative values), and/or leading zeros to numbers in a given column. You can also round the values to a given number of decimal places, display minus signs to the right of negative values (instead of to the left), and display values in exponential notation.

To use more than one format model for a single column, combine the desired models in one COLUMN command (see Example 4–4). For a complete list of format models and further details, see the COLUMN command in Chapter 7.

Example 4–4 Formatting a NUMBER Column

To display SAL with a dollar sign, a comma, and the numeral zero instead of a blank for any zero values, enter the following command:

```
SQL> COLUMN SAL FORMAT $99,990
```

Now re-run the current query:

```
SQL> /
```

SQL*Plus displays the following output:

```
          Employee
Department Name          Salary Commission
---------- ---------- ---------- ----------
        30  ALLEN       $1,600        300
        30  WARD        $1,250        500
        30  MARTIN      $1,250       1400
        30  TURNER      $1,500          0
```

Use a zero in your format model, as shown above, when you use other formats such as a dollar sign and wish to display a zero in place of a blank for zero values.

> **Note:** The format model will stay in effect until you enter a new one, reset the column's format, or exit from SQL*Plus.

Formatting Datatypes and Trusted Oracle Columns

When displaying datatypes and Trusted Oracle columns, you can either accept the SQL*Plus default display width or you can change it using the COLUMN command. Datatypes, in this manual, include the following variables:

- CHAR
- NCHAR
- VARCHAR2 (VARCHAR)
- NVARCHAR2 (NCHAR VARYING)
- DATE
- LONG
- CLOB
- NCLOB

> **Note:** The NCHAR, NVARCHAR2 (NCHAR VARYING), CLOB and NCLOB datatypes require Oracle8 or higher.

The sections below describe the defaults and how you can alter the defaults with the COLUMN command.

Default Display

The default width of datatype columns is the width of the column in the database.

The display width of LONG, CLOB and NCLOB columns defaults to the value of the LONG or LONGCHUNKSIZE, whichever is smaller. See the SET command in chapter 7 for information about the LONG and LONGCHUNKSIZE variables.

The default width and format of unformatted DATE columns in SQL*Plus is derived from the NLS parameters in effect. Otherwise, the default format width is A9. For more information on formatting DATE columns, see the FORMAT clause of the COLUMN command in Chapter 7.

The default display width for the Trusted Oracle datatype MLSLABEL is the width defined for the column in the database or the width of the column heading, whichever is longer. (Note that the default display width for a Trusted Oracle column named ROWLABEL is 15.)

> **Note:** The default justification for datatypes and Trusted Oracle columns is left justification.

Changing the Default Display

You can change the displayed width of a datatype, DATE, or Trusted Oracle column by using the COLUMN command with a format model consisting of the letter A (for alphanumeric) followed by a number representing the width of the column in characters.

Within the COLUMN command, identify the column you want to format and the model you want to use:

```
COLUMN column_name FORMAT model
```

If you specify a width shorter than the column heading, SQL*Plus truncates the heading. If you specify a width for a LONG, CLOB, or NCLOB column, SQL*Plus uses the LONGCHUNKSIZE or the specified width, whichever is smaller, as the column width. See the COLUMN command in Chapter 7 for more details.

Example 4–5 Formatting a Character Column

To set the width of the column ENAME to four characters and re-run the current query, enter

```
SQL> COLUMN ENAME FORMAT A4
SQL> /
```

SQL*Plus displays the results:

```
           Empl
Department Name     Salary Commission
---------- ---- ---------- ----------
        30  ALLE    $1,600        300
            N

        30  WARD    $1,250        500
        30  MART    $1,250       1400
            IN

        30  TURN    $1,500          0
            ER
```

> **Note:** The format model will stay in effect until you enter a new one, reset the column's format, or exit from SQL*Plus. ENAME could be a CHAR, NCHAR, VARCHAR2 (VARCHAR), or NVARCHAR2 (NCHAR VARYING) column.

If the WRAP variable of the SET command is set to ON (its default value), the employee names wrap to the next line after the fourth character, as shown in Example 4–5. If WRAP is set to OFF, the names are truncated (cut off) after the fourth character.

The system variable WRAP controls all columns; you can override the setting of WRAP for a given column through the WRAPPED, WORD_WRAPPED, and TRUNCATED clauses of the COLUMN command. See COLUMN in Chapter 7 for more information on these clauses. You will use the WORD_WRAPPED clause of COLUMN later in this chapter.

> **Note:** The column heading is truncated regardless of the setting of WRAP or any COLUMN command clauses.

Now return the column to its previous format:

```
SQL> COLUMN ENAME FORMAT A10
```

Copying Column Display Attributes

When you want to give more than one column the same display attributes, you can reduce the length of the commands you must enter by using the LIKE clause of the COLUMN command. The LIKE clause tells SQL*Plus to copy the display attributes of a previously defined column to the new column, except for changes made by other clauses in the same command.

Example 4–6
Copying a Column's
Display Attributes

To give the column COMM the same display attributes you gave to SAL, but to specify a different heading, enter the following command:

```
SQL> COLUMN COMM LIKE SAL HEADING Bonus
```

Re-run the query:

```
SQL> /
```

SQL*Plus displays the following output:

```
          Employee
Department Name          Salary      Bonus
---------- ---------- ---------- ----------
        30 ALLEN          $1,600       $300
        30 WARD           $1,250       $500
        30 MARTIN         $1,250     $1,400
        30 TURNER         $1,500         $0
```

Listing and Resetting Column Display Attributes

To list the current display attributes for a given column, use the COLUMN command followed by the column name only, as shown below:

```
COLUMN column_name
```

To list the current display attributes for all columns, enter the COLUMN command with no column names or clauses after it:

```
COLUMN
```

To reset the display attributes for a column to their default values, use the CLEAR clause of the COLUMN command as shown below:

```
COLUMN column_name CLEAR
```

To reset the attributes for all columns, use the COLUMNS clause of the CLEAR command.

Example 4–7
Resetting Column
Display Attributes to
their Defaults

To reset all columns' display attributes to their default values, enter the following command:

```
SQL> CLEAR COLUMNS
columns cleared
```

You may wish to place the command CLEAR COLUMNS at the beginning of every command file to ensure that previously entered COLUMN commands will not affect queries you run in a given file.

Suppressing and Restoring Column Display Attributes

You can suppress and restore the display attributes you have given a specific column. To suppress a column's display attributes, enter a COLUMN command in the following form:

```
COLUMN column_name OFF
```

The OFF clause tells SQL*Plus to use the default display attributes for the column, but does not remove the attributes you have defined through the COLUMN command. To restore the attributes you defined through COLUMN, use the ON clause:

```
COLUMN column_name ON
```

Printing a Line of Characters after Wrapped Column Values

As you have seen, by default SQL*Plus wraps column values to additional lines when the value does not fit within the column width. If you want to insert a *record separator* (a line of characters or a blank line) after each wrapped line of output (or after every row), use the RECSEP and RECSEPCHAR variables of the SET command.

RECSEP determines when the line of characters is printed; you set RECSEP to EACH to print after every line, to WRAPPED to print after wrapped lines, and to OFF to suppress printing. The default setting of RECSEP is WRAPPED.

RECSEPCHAR sets the character printed in each line. You can set RECSEPCHAR to any character.

You may wish to wrap whole words to additional lines when a column value wraps to additional lines. To do so, use the WORD_WRAPPED clause of the COLUMN command as shown below:

```
COLUMN column_name WORD_WRAPPED
```

Example 4–8 Printing a Line of Characters after Wrapped Column Values

To print a line of dashes after each wrapped column value, enter the following commands:

```
SQL> SET RECSEP WRAPPED
SQL> SET RECSEPCHAR '-'
```

Now restrict the width of the column LOC and tell SQL*Plus to wrap whole words to additional lines when necessary:

```
SQL> COLUMN LOC FORMAT A7 WORD_WRAPPED
```

Finally, enter and run the following query:

```
SQL> SELECT * FROM DEPT;
```

SQL*Plus displays the results:

```
    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 ACCOUNTING     NEW
                          YORK
--------------------------------------------------
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

If you set RECSEP to EACH, SQL*Plus prints a line of characters after every row (after every department, for the above example).

Before continuing, set RECSEP to OFF to suppress the printing of record separators:

```
SQL> SET RECSEP OFF
```

Clarifying Your Report with Spacing and Summary Lines

When you use an ORDER BY clause in your SQL SELECT command, rows with the same value in the ordered column (or expression) are displayed together in your output. You can make this output more useful to the user by using the SQL*Plus BREAK and COMPUTE commands to create subsets of records and add space and/or summary lines after each subset.

The column you specify in a BREAK command is called a *break column.* By including the break column in your ORDER BY clause, you create meaningful subsets of records in your output. You can then add formatting to the subsets within the same BREAK command, and add a summary line (containing totals, averages, and so on) by specifying the break column in a COMPUTE command.

For example, the following query, without BREAK or COMPUTE commands,

```
SQL> SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE SAL < 2500
  4  ORDER BY DEPTNO;
```

produces the following unformatted results:

```
DEPTNO  ENAME           SAL
--------  ----------  ---------
     10  CLARK          2450
     10  MILLER         1300
     20  SMITH           800
     20  ADAMS          1100
     30  ALLEN          1600
     30  JAMES           950
     30  TURNER         1500
     30  WARD           1250
     30  MARTIN         1250
```

To make this report more useful, you would use BREAK to establish DEPTNO as the break column. Through BREAK you could suppress duplicate values in DEPTNO and place blank lines or begin a new page between departments. You could use BREAK in conjunction with COMPUTE to calculate and print summary lines containing the total (and/or average, maximum, minimum, standard deviation, variance, or count of rows of) salary for each department and for all departments.

Suppressing Duplicate Values in Break Columns

The BREAK command suppresses duplicate values by default in the column or expression you name. Thus, to suppress the duplicate values in a column specified in an ORDER BY clause, use the BREAK command in its simplest form:

```
BREAK ON break_column
```

> **Note:** Whenever you specify a column or expression in a BREAK command, use an ORDER BY clause specifying the same column or expression. If you do not do this, the breaks may appear to occur randomly.

Example 4–9 Suppressing Duplicate Values in a Break Column

To suppress the display of duplicate department numbers in the query results shown above, enter the following commands:

```
SQL> BREAK ON DEPTNO
SQL> SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE SAL < 2500
  4  ORDER BY DEPTNO;
```

SQL*Pus displays the following output:

```
    DEPTNO ENAME             SAL
---------- ----------- ---------
        10 CLARK            2450
           MILLER           1300
        20 SMITH             800
           ADAMS            1100
        30 ALLEN            1600
           JAMES             950
           TURNER           1500
           WARD             1250
           MARTIN           1250
```

Inserting Space when a Break Column's Value Changes

You can insert blank lines or begin a new page each time the value changes in the break column. To insert *n* blank lines, use the BREAK command in the following form:

```
BREAK ON break_column SKIP n
```

To skip a page, use the command in this form:

```
BREAK ON break_column SKIP PAGE
```

Example 4–10 Inserting Space when a Break Column's Value Changes

To place one blank line between departments, enter the following command:

```
SQL> BREAK ON DEPTNO SKIP 1
```

Now re-run the query:

```
SQL> /
```

SQL*Plus displays the results:

```
    DEPTNO ENAME             SAL
---------- ----------- ---------
        10 CLARK            2450
           MILLER           1300

        20 SMITH             800
           ADAMS            1100

        30 ALLEN            1600
           JAMES             950
           TURNER           1500
           WARD             1250
           MARTIN           1250
```

Inserting Space after Every Row

You may wish to insert blank lines or a blank page after every row. To skip *n* lines after every row, use BREAK in the following form:

```
BREAK ON ROW SKIP n
```

To skip a page after every row, use

```
BREAK ON ROW SKIP PAGE
```

> **Note:** SKIP PAGE does not cause a physical page break unless you have also specified NEWPAGE 0.

Using Multiple Spacing Techniques

Suppose you have more than one column in your ORDER BY clause and wish to insert space when each column's value changes. Each BREAK command you enter replaces the previous one. Thus, if you want to use different spacing techniques in one report or insert space after the value changes in more than one ordered column, you must specify multiple columns and actions in a single BREAK command.

Example 4–11 Combining Spacing Techniques

First, add another column to the current query:

```
SQL> L
  1  SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE SAL < 2500
  4* ORDER BY DEPTNO
SQL> 1 SELECT DEPTNO, JOB, ENAME, SAL
SQL> 4 ORDER BY DEPTNO, JOB
```

Now, to skip a page when the value of DEPTNO changes and one line when the value of JOB changes, enter the following command:

```
SQL> BREAK ON DEPTNO SKIP PAGE ON JOB SKIP 1
```

To show that SKIP PAGE has taken effect, create a TTITLE with a page number, enter

```
SQL> TTITLE COL 35 FORMAT 9 'Page:' SQL.PNO
```

Run the new query to see the results:

```
SQL> /

                                  Page: 1
    DEPTNO JOB        ENAME             SAL
---------- --------- ---------- ----------
        10 CLERK     MILLER            300

           MANAGER   CLARK            2450
```

```
                                           Page: 2
            DEPTNO JOB       ENAME             SAL

        ---------- --------- ---------- ----------
                20 CLERK     SMITH              800
                             ADAMS             1100


                                           Page: 3
            DEPTNO JOB       ENAME             SAL

        ---------- --------- ---------- ----------
                30 CLERK     JAMES              950

                   SALESMAN  ALLEN             1600
                             TURNER            1500
                             WARD              1250
                             MARTIN            1250
```

Listing and Removing Break Definitions

You can list your current break definition by entering the BREAK command with no clauses:

```
BREAK
```

You can remove the current break definition by entering the CLEAR command with the BREAKS clause:

```
CLEAR BREAKS
```

You may wish to place the command CLEAR BREAKS at the beginning of every command file to ensure that previously entered BREAK commands will not affect queries you run in a given file.

Computing Summary Lines when a Break Column's Value Changes

If you organize the rows of a report into subsets with the BREAK command, you can perform various computations on the rows in each subset. You do this with the functions of the SQL*Plus COMPUTE command. Use the BREAK and COMPUTE commands together in the following forms:

```
BREAK ON break_column
COMPUTE function LABEL label_name OF column column column
... ON break_column
```

You can include multiple break columns and actions, such as skipping lines in the BREAK command, as long as the column you name after ON in the COMPUTE command also appears after ON in the BREAK command. To include multiple break columns and actions in BREAK when using it in conjunction with COMPUTE, use these commands in the following forms:

```
BREAK ON break_column_1 SKIP PAGE ON break_column_2 SKIP 1
COMPUTE function LABEL label_name OF column column column
... ON break_column_2
```

The COMPUTE command has no effect without a corresponding
BREAK command.

You can COMPUTE on NUMBER columns and, in certain cases, on all
types of columns. See COMPUTE in Chapter 7 for details.

The following table lists compute functions and their effects:

Function	*Effect*
SUM	Computes the sum of the values in the column.
MINIMUM	Computes the minimum value in the column.
MAXIMUM	Computes the maximum value in the column.
AVG	Computes the average of the values in the column.
STD	Computes the standard deviation of the values in the column.
VARIANCE	Computes the variance of the values in the column.
COUNT	Computes the number of non-null values in the column.
NUMBER	Computes the number of rows in the column.

Table 4 – 1 Compute Functions

The function you specify in the COMPUTE command applies to all
columns you enter after OFF and before ON. The computed values print
on a separate line when the value of the ordered column changes.

Labels for ON REPORT and ON ROW computations appear in the first
column; otherwise, they appear in the column specified in the ON
clause.

You can change the compute label by using COMPUTE LABEL. If you
do not define a label for the computed value, SQL*Plus prints the
unabbreviated function keyword.

The compute label can be suppressed by using the NOPRINT option of
the COLUMN command on the break column. See the COMPUTE
command in Chapter 7 for more details.

Example 4–12 Computing and Printing Subtotals

To compute the total of SAL by department, first list the current BREAK definition:

```
SQL> BREAK
break on DEPTNO skip 0 page nodup
          on JOB skip 1 nodup
```

Now enter the following COMPUTE command and run the current query:

```
SQL> COMPUTE SUM OF SAL ON DEPTNO
SQL> /
```

SQL*Plus displays the following output:

```
    DEPTNO JOB       ENAME            SAL
---------- --------- ---------- ----------
        10 CLERK     MILLER          1300

           MANAGER   CLARK           2450
********** *********            ----------
sum                                 3750

    DEPTNO JOB       ENAME            SAL
---------- --------- ---------- ----------
        20 CLERK     SMITH            800
                     ADAMS           1100
********** *********            ----------
sum                                 1900

    DEPTNO JOB       ENAME            SAL
---------- --------- ---------- ----------
        30 CLERK     JAMES            950

           SALESMAN  ALLEN           1600
                     TURNER          1500
                     WARD            1250
                     MARTIN          1250
********** *********            ----------
sum                                 6550
```

To compute the sum of salaries for departments 10 and 20 without printing the compute label:

```
SQL> COLUMN DUMMY NOPRINT
SQL> COMPUTE SUM OF SAL ON DUMMY
SQL> BREAK ON DUMMY SKIP 1
SQL> SELECT DEPTNO DUMMY, DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO <= 20
  4  ORDER BY DEPTNO;
```

SQL*Plus displays the following output:

```
    DEPTNO ENAME              SAL
---------- ---------- ----------
        10 KING             5000
        10 CLARK            2450
        10 MILLER           1300
                      ----------
                            8750

        20 JONES            2975
        20 FORD             3000
        20 SMITH             800
        20 SCOTT            3000
        20 ADAMS            1100
                      ----------
                           10875
```

To compute the salaries at the end of the report:

```
SQL> COLUMN DUMMY NOPRINT
SQL> COMPUTE SUM OF SAL ON DUMMY
SQL> BREAK ON DUMMY
SQL> SELECT NULL DUMMY, DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO <= 20
  4  ORDER BY DEPTNO;
```

SQL*Plus displays the following output:

```
    DEPTNO ENAME             SAL
---------- ---------- ----------
        10 KING             5000
        10 CLARK            2450
        10 MILLER           1300
        20 JONES            2975
        20 FORD             3000
        20 SMITH             800
        20 SCOTT            3000
        20 ADAMS            1100
                      ----------
                           19625
```

> **Note:** The format of the column SAL controls the appearance of the sum of SAL, as well as the individual values of SAL. When you establish the format of a NUMBER column, you must allow for the size of sums you will include in your report.

Computing Summary Lines at the End of the Report

You can calculate and print summary lines based on all values in a column by using BREAK and COMPUTE in the following forms:

```
BREAK ON REPORT
COMPUTE function LABEL label_name OF column column column
... ON REPORT
```

Example 4–13
Computing and Printing a Grand Total

To calculate and print the grand total of salaries for all salesmen and change the compute label, first enter the following BREAK and COMPUTE commands:

```
SQL> BREAK ON REPORT
SQL> COMPUTE SUM LABEL TOTAL OF SAL ON REPORT
```

Next, enter and run a new query:

```
SQL> SELECT ENAME, SAL
  2  FROM EMP
  3  WHERE JOB = 'SALESMAN';
```

SQL*Plus displays the results:

```
ENAME           SAL
---------- --------
ALLEN          1600
WARD           1250
MARTIN         1250
TURNER         1500
********** --------
TOTAL          5600
```

To print a grand total (or grand average, grand maximum, and so on) in addition to subtotals (or sub-averages, and so on), include a break column and an ON REPORT clause in your BREAK command. Then, enter one COMPUTE command for the break column and another to compute ON REPORT:

```
BREAK ON break_column ON REPORT
COMPUTE function LABEL label_name OF column ON break_column
COMPUTE function LABEL label_name OF column ON REPORT
```

Computing Multiple Summary Values and Lines

You can compute and print the same type of summary value on different columns. To do so, enter a separate COMPUTE command for each column.

Example 4–14 Computing the Same Type of Summary Value on Different Columns

To print the total of salaries and commissions for all salesmen, first enter the following COMPUTE command:

```
SQL> COMPUTE SUM OF SAL COMM ON REPORT
```

You do not have to enter a BREAK command; the BREAK you entered in Example 4–13 is still in effect. Now, add COMM to the current query:

```
SQL> 1 SELECT ENAME, SAL, COMM
```

Finally, run the revised query to see the results:

```
SQL> /

ENAME           SAL       COMM
---------- -------- ----------
ALLEN          1600        300
WARD           1250        500
MARTIN         1250       1400
TURNER         1500          0
********** -------- ----------
sum            5600       2200
```

You can also print multiple summary lines on the same break column. To do so, include the function for each summary line in the COMPUTE command as follows:

```
COMPUTE function LABEL label_name function
   LABEL label_name function LABEL label_name ...
   OF column ON break_column
```

If you include multiple columns after OFF and before ON, COMPUTE calculates and prints values for each column you specify.

Example 4–15
Computing Multiple
Summary Lines on the
Same Break Column

To compute the average and sum of salaries for the sales department, first enter the following BREAK and COMPUTE commands:

```
SQL> BREAK ON DEPTNO
SQL> COMPUTE AVG SUM OF SAL ON DEPTNO
```

Now, enter and run the following query:

```
SQL> SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO = 30
  4  ORDER BY DEPTNO, SAL;
```

SQL*Plus displays the results:

```
    DEPTNO ENAME            SAL
---------- ---------- ----------
        30 JAMES            950
           WARD            1250
           MARTIN          1250
           TURNER          1500
           ALLEN           1600
           BLAKE           2850
**********            ----------
avg                   1566.66667
sum                         9400
```

Listing and Removing
COMPUTE Definitions

You can list your current COMPUTE definitions by entering the COMPUTE command with no clauses:

```
COMPUTE
```

You can remove all the COMPUTE definitions by entering the CLEAR command with the COMPUTES clause.

Example 4–16
Removing COMPUTE
Definitions

To remove all COMPUTE definitions and the accompanying BREAK definition, enter the following commands:

```
SQL> CLEAR BREAKS
breaks cleared
SQL> CLEAR COMPUTES
computes cleared
```

You may wish to place the commands CLEAR BREAKS and CLEAR COMPUTES at the beginning of every command file to ensure that previously entered BREAK and COMPUTE commands will not affect queries you run in a given file.

Defining Page and Report Titles and Dimensions

The word *page* refers to a screenful of information on your display or a page of a spooled (printed) report. You can place top and bottom titles on each page, set the number of lines per page, and determine the width of each line.

The word *report* refers to the complete results of a query. You can also place headers and footers on each report and format them in the same way as top and bottom titles on pages.

Setting the Top and
Bottom Titles and
Headers and Footers

As you have already seen, you can set a title to display at the top of each page of a report. You can also set a title to display at the bottom of each page. The TTITLE command defines the top title; the BTITLE command defines the bottom title.

You can also set a header and footer for each report. The REPHEADER command defines the report header; the REPFOOTER command defines the report footer.

A TTITLE, BTITLE, REPHEADER or REPFOOTER command consists of the command name followed by one or more clauses specifying a position or format and a CHAR value you wish to place in that position or give that format. You can include multiple sets of clauses and CHAR values:

```
TTITLE position_clause(s) char_value position_clause(s)
char_value ...
BTITLE position_clause(s) char_value position_clause(s)
char_value ...
REPHEADER position_clause(s) char_value position_clause(s)
char_value ...
REPFOOTER position_clause(s) char_value position_clause(s)
char_value ...
```

The most often used clauses of TTITLE, BTITLE, REPHEADER and REPFOOTER are summarized in the following table. For descriptions of all TTITLE, BTITLE, REPHEADER and REPFOOTER clauses, see the discussions of TTITLE and REPHEADER in Chapter 7.

Clause	*Example*	*Description*
COL *n*	COL 72	Makes the next CHAR value appear in the specified column of the line.
SKIP *n*	SKIP 2	Skips to a new line *n* times. If *n* is greater than 1, *n*–1 blank lines appear before the next CHAR value.
LEFT	LEFT	Left-aligns the following CHAR value.
CENTER	CENTER	Centers the following CHAR value.
RIGHT	RIGHT	Right-aligns the following CHAR value.

Table 4 – 2 Often-Used Clauses of TTITLE, BTITLE, REPHEADER and REPFOOTER

**Example 4–17
Placing a Top and
Bottom Title on a Page**   To put titles at the top and bottom of each page of a report, enter

```
SQL> TTITLE CENTER –
>    'ACME WIDGET SALES DEPARTMENT PERSONNEL REPORT'
SQL> BTITLE CENTER 'COMPANY CONFIDENTIAL'
```

Now run the current query:

```
SQL> /
```

SQL*Plus displays the following output:

```
             ACME WIDGET SALES DEPARTMENT PERSONNEL REPORT
    DEPTNO ENAME             SAL
---------- ---------- ----------
        30 JAMES             950
        30 WARD             1250
        30 MARTIN           1250
        30 TURNER           1500
        30 ALLEN            1600
        30 BLAKE            2850



                        COMPANY CONFIDENTIAL
```

Example 4–18
Placing a Header on a Report

To put a report header on a separate page, and to center it, enter

```
SQL> REPHEADER PAGE CENTER 'ACME WIDGET'
```

Now run the current query:

```
SQL> /
```

SQL*Plus displays the following output on page one

```
            ACME WIDGET SALES DEPARTMENT PERSONNEL REPORT
                            ACME WIDGET






                        COMPANY CONFIDENTIAL
```

and the following output on page two

```
              ACME WIDGET SALES DEPARTMENT PERSONNEL REPORT
    DEPTNO ENAME           SAL
---------- ---------- ----------
        30 JAMES             950
        30 WARD             1250
        30 MARTIN           1250
        30 TURNER           1500
        30 ALLEN            1600
        30 BLAKE            2850




                         COMPANY CONFIDENTIAL
```

To suppress the report header without changing its definition, enter

```
SQL> REPHEADER OFF
```

Positioning Title Elements   The report in the preceding exercises might look more attractive if you give the company name more emphasis and place the type of report and the department name on either end of a separate line. It may also help to reduce the linesize and thus center the titles more closely around the data.

You can accomplish these changes by adding some clauses to the TTITLE command and by resetting the system variable LINESIZE, as the following example shows.

You can format report headers and footers in the same way as BTITLE and TTITLE using the REPHEADER and REPFOOTER commands.

Example 4–19
Positioning Title
Elements   To redisplay the personnel report with a repositioned top title, enter the following commands:

```
SQL> TTITLE CENTER 'A C M E   W I D G E T' SKIP 1 –
>  CENTER =============== SKIP 1 LEFT 'PERSONNEL REPORT' –
>  RIGHT 'SALES DEPARTMENT' SKIP 2
SQL> SET LINESIZE 60
SQL> /
```

SQL*Plus displays the results:

```
                    A C M E   W I D G E T
                    ====================
PERSONNEL REPORT                              SALES DEPARTMENT

    DEPTNO ENAME             SAL
---------- ---------- ----------
        30 JAMES              950
        30 WARD              1250
        30 MARTIN            1250
        30 TURNER            1500
        30 ALLEN             1600
        30 BLAKE             2850
                    COMPANY CONFIDENTIAL
```

The LEFT, RIGHT, and CENTER clauses place the following values at the beginning, end, and center of the line. The SKIP clause tells SQL*Plus to move down one or more lines.

Note that there is no longer any space between the last row of the results and the bottom title. The last line of the bottom title prints on the last line of the page. The amount of space between the last row of the report and the bottom title depends on the overall page size, the number of lines occupied by the top title, and the number of rows in a given page. In the above example, the top title occupies three more lines than the top title in the previous example. You will learn to set the number of lines per page later in this chapter.

To always print *n* blank lines before the bottom title, use the SKIP *n* clause at the beginning of the BTITLE command. For example, to skip one line before the bottom title in the example above, you could enter the following command:

```
BTITLE SKIP 1 CENTER 'COMPANY CONFIDENTIAL'
```

Indenting a Title Element You can use the COL clause in TTITLE or BTITLE to indent the title element a specific number of spaces. For example, COL 1 places the following values in the first character position, and so is equivalent to LEFT, or an indent of zero. COL 15 places the title element in the 15th character position, indenting it 14 spaces.

To print the company name left-aligned with the report name indented five spaces on the next line, enter

```
SQL> TTITLE LEFT 'ACME WIDGET' SKIP 1 –
>  COL 6 'SALES DEPARTMENT PERSONNEL REPORT' SKIP 2
```

Now re-run the current query to see the results:

```
SQL> /
ACME WIDGET
     SALES DEPARTMENT PERSONNEL REPORT


    DEPTNO ENAME             SAL
---------- ---------- ----------
        30 JAMES             950
        30 WARD             1250
        30 MARTIN           1250
        30 TURNER           1500
        30 ALLEN            1600
        30 BLAKE            2850


                    COMPANY CONFIDENTIAL
```

Entering Long Titles

If you need to enter a title greater than 500 characters in length, you can use the SQL*Plus command DEFINE to place the text of each line of the title in a separate user variable:

```
SQL> DEFINE LINE1 = 'This is the first line...'
SQL> DEFINE LINE2 = 'This is the second line...'
SQL> DEFINE LINE3 = 'This is the third line...'
```

Then, reference the variables in your TTITLE or BTITLE command as follows:

```
SQL> TTITLE CENTER LINE1 SKIP 1 CENTER LINE2 SKIP 1 –
>    CENTER LINE3
```

Displaying the Page Number and other System-Maintained Values in Titles

You can display the current page number and other system-maintained values in your title by entering a system value name as a title element, for example:

```
TTITLE LEFT system-maintained_value_name
```

There are five system-maintained values you can display in titles, the most commonly used of which is SQL.PNO (the current page number). Refer to the TTITLE command in Chapter 7 for a list of system-maintained values you can display in titles.

**Example 4–21
Displaying the Current
Page Number in a Title**

To display the current page number at the top of each page, along with the company name, enter the following command:

```
SQL> TTITLE LEFT 'ACME WIDGET' RIGHT 'PAGE:' SQL.PNO SKIP 2
```

Now re-run the current query:

```
SQL> /
```

SQL*Plus displays the following results:

```
ACME WIDGET                                        PAGE:          1

    DEPTNO ENAME            SAL
---------- ---------- ----------
        30 JAMES            950
        30 WARD            1250
        30 MARTIN          1250
        30 TURNER          1500
        30 ALLEN           1600
        30 BLAKE           2850



                        COMPANY CONFIDENTIAL
```

Note that SQL.PNO has a format ten spaces wide. You can change this format with the FORMAT clause of TTITLE (or BTITLE).

**Example 4–22
Formatting a
System-Maintained
Value in a Title**

To close up the space between the word PAGE: and the page number, re-enter the TTITLE command as shown:

```
SQL> TTITLE LEFT 'ACME WIDGET' RIGHT 'PAGE:' FORMAT 999 -
>  SQL.PNO SKIP 2
```

Now re-run the query:

```
SQL> /
```

SQL*Plus displays the following results:

```
ACME WIDGET                                          PAGE:  1

    DEPTNO ENAME             SAL
---------- ---------- ----------
        30 JAMES             950
        30 WARD             1250
        30 MARTIN           1250
        30 TURNER           1500
        30 ALLEN            1600
        30 BLAKE            2850


                    COMPANY CONFIDENTIAL
```

Listing, Suppressing, and Restoring Page Title Definitions

To list a page title definition, enter the appropriate title command with no clauses:

```
TTITLE
BTITLE
```

To suppress a title definition, enter:

```
TTITLE OFF
BTITLE OFF
```

These commands cause SQL*Plus to cease displaying titles on reports, but do not clear the current definitions of the titles. You may restore the current definitions by entering

```
TTITLE ON
BTITLE ON
```

Displaying Column Values in Titles

You may wish to create a master/detail report that displays a changing master column value at the top of each page with the detail query results for that value below. You can reference a column value in a top title by storing the desired value in a variable and referencing the variable in a TTITLE command. Use the following form of the COLUMN command to define the variable:

```
COLUMN column_name NEW_VALUE variable_name
```

You must include the master column in an ORDER BY clause and in a BREAK command using the SKIP PAGE clause.

**Example 4–23
Creating a
Master/Detail Report**

Suppose you want to create a report that displays two different managers' employee numbers, each at the top of a separate page, and the people reporting to the manager on the same page as the manager's employee number. First create a variable, MGRVAR, to hold the value of the current manager's employee number:

```
SQL> COLUMN MGR NEW_VALUE MGRVAR NOPRINT
```

Because you will display the managers' employee numbers in the title, you do not want them to print as part of the detail. The NOPRINT clause you entered above tells SQL*Plus not to print the column MGR.

Next, include a label and the value in your page title, enter the proper BREAK command, and suppress the bottom title from the last example:

```
SQL> TTITLE LEFT 'Manager: ' MGRVAR SKIP 2
SQL> BREAK ON MGR SKIP PAGE
SQL> BTITLE OFF
```

Finally, enter and run the following query:

```
SQL> SELECT MGR, ENAME, SAL, DEPTNO
  2  FROM EMP
  3  WHERE MGR IN (7698, 7839)
  3  ORDER BY MGR;
```

SQL*Plus displays the following output:

```
Manager:      7698


ENAME           SAL     DEPTNO
---------- -------- ----------
ALLEN          1600         30
WARD           1250         30
TURNER         1500         30
MARTIN         1250         30
JAMES           950         30


Manager:      7839


ENAME           SAL     DEPTNO
---------- -------- ----------
JONES          2975         20
BLAKE          2850         30
CLARK          2450         10
```

If you want to print the value of a column at the bottom of the page, you can use the COLUMN command in the following form:

```
COLUMN column_name OLD_VALUE variable_name
```

SQL*Plus prints the bottom title as part of the process of breaking to a new page—after finding the new value for the master column. Therefore, if you simply referenced the NEW_VALUE of the master column, you would get the value for the next set of details. OLD_VALUE remembers the value of the master column that was in effect before the page break began.

Displaying the Current Date in Titles

You can, of course, date your reports by simply typing a value in the title. This is satisfactory for ad hoc reports, but if you want to run the same report repeatedly, you would probably prefer to have the date automatically appear when the report is run. You can do this by creating a variable to hold the current date.

To create the variable (in this example named _DATE), you can add the following commands to your SQL*Plus LOGIN file:

```
SET TERMOUT OFF
BREAK ON TODAY
COLUMN TODAY NEW_VALUE _DATE
SELECT TO_CHAR(SYSDATE, 'fmMonth DD, YYYY') TODAY
FROM DUAL;
CLEAR BREAKS
SET TERMOUT ON
```

When you start SQL*Plus, these commands place the value of SYSDATE (the current date) into a variable named _DATE. To display the current date, you can reference _DATE in a title as you would any other variable.

The date format model you include in the SELECT command in your LOGIN file determines the format in which SQL*Plus displays the date. See your *Oracle8 Server SQL Reference Manual* for more information on date format models. For more information about the LOGIN file, see "Modifying Your LOGIN File" in Chapter 3.

You can also enter these commands interactively at the command prompt. For more information, see the COLUMN command in Chapter 7.

Setting Page Dimensions

Typically, a page of a report contains the number of blank line(s) set in the NEWPAGE variable of the SET command, a top title, column headings, your query results, and a bottom title. SQL*Plus displays a report that is too long to fit on one page on several consecutive pages,

each with its own titles and column headings. The amount of data SQL*Plus displays on each page depends on the current page dimensions.

The default page dimensions used by SQL*Plus are shown below:

- number of lines before the top title: 1
- number of lines per page, from the top title to the bottom of the page: 24
- number of characters per line: 80

You can change these settings to match the size of your computer screen or, for printing, the size of a sheet of paper.

You can change the page length with the system variable PAGESIZE. For example, you may wish to do so when you print a report, since printed pages are customarily 66 lines long.

To set the number of lines between the beginning of each page and the top title, use the NEWPAGE variable of the SET command:

```
SET NEWPAGE number_of_lines
```

If you set NEWPAGE to zero, SQL*Plus skips zero lines and displays and prints a formfeed character to begin a new page. On most types of computer screens, the formfeed character clears the screen and moves the cursor to the beginning of the first line. When you print a report, the formfeed character makes the printer move to the top of a new sheet of paper, even if the overall page length is less than that of the paper. If you set NEWPAGE to NONE, SQL*Plus does not print a blank line or formfeed between report pages.

To set the number of lines on a page, use the PAGESIZE variable of the SET command:

```
SET PAGESIZE number_of_lines
```

You may wish to reduce the linesize to center a title properly over your output, or you may want to increase linesize for printing on wide paper. You can change the line width using the LINESIZE variable of the SET command:

```
SET LINESIZE number_of_characters
```

**Example 4–24
Setting Page
Dimensions**

To set the page size to 66 lines, clear the screen (or advance the printer to a new sheet of paper) at the start of each page, and set the linesize to 32, enter the following commands:

```
SQL> SET PAGESIZE 66
SQL> SET NEWPAGE 0
SQL> SET LINESIZE 32
```

Now enter and run the following commands to see the results:

```
SQL> TTITLE CENTER 'ACME WIDGET PERSONNEL REPORT' SKIP 1 -
> CENTER '10-JAN-89' SKIP 2
SQL> COLUMN DEPTNO HEADING DEPARTMENT
SQL> COLUMN ENAME HEADING EMPLOYEE
SQL> COLUMN SAL FORMAT $99,999 HEADING SALARY
SQL> SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  ORDER BY DEPTNO;
```

SQL*Plus displays a formfeed followed by the query results:

```
  ACME WIDGET PERSONNEL REPORT
            10-JAN-89


DEPARTMENT EMPLOYEE      SALARY
---------- ---------- ----------
        10 CLARK         $2,450
        10 KING          $5,000
        10 MILLER        $1,300
        20 SMITH           $800
        20 ADAMS         $1,100
        20 FORD          $3,000
        20 SCOTT         $3,000
        20 JONES         $2,975
        30 ALLEN         $1,600
        30 BLAKE         $2,850
        30 MARTIN        $1,250
        30 JAMES           $950
        30 TURNER        $1,500
        30 WARD          $1,250
```

Now reset PAGESIZE, NEWPAGE, and LINESIZE to their default values:

```
SQL> SET PAGESIZE 24
SQL> SET NEWPAGE 1
SQL> SET LINESIZE 80
```

To list the current values of these variables, use the SHOW command:

```
SQL> SHOW PAGESIZE
pagesize 24
SQL> SHOW NEWPAGE
newpage 1
SQL> SHOW LINESIZE
linesize 80
```

Through the SQL*Plus command SPOOL, you can store your query results in a file or print them on your computer's default printer.

Sending Results to a File

To store the results of a query in a file—and still display them on the screen—enter the SPOOL command in the following form:

```
SPOOL file_name
```

SQL*Plus stores all information displayed on the screen after you enter the SPOOL command in the file you specify.

Storing and Printing Query Results

Send your query results to a file when you want to edit them with a word processor before printing or include them in a letter, memo, or other document.

To store the results of a query in a file—and still display them on the screen—enter the SPOOL command in the following form:

```
SPOOL file_name
```

If you do not follow the filename with a period and an extension, SPOOL adds a default file extension to the filename to identify it as an output file. The default varies with the host operating system; on most hosts it is LST or LIS. See the Oracle installation and user's manual(s) provided for your operating system for more information.

SQL*Plus continues to spool information to the file until you turn spooling off, using the following form of SPOOL:

```
SPOOL OFF
```

Creating a Flat File

When moving data between different software products, it is sometimes necessary to use a "flat" file (an operating system file with no escape characters, headings, or extra characters embedded). For example, if you do not have SQL*Net, you need to create a flat file for use with SQL*Loader when moving data from Oracle7 to Oracle8.

To create a flat file with SQL*Plus, you first must enter the following SET commands:

```
SET NEWPAGE 0
SET SPACE 0
SET LINESIZE 80
SET PAGESIZE 0
SET ECHO OFF
SET FEEDBACK OFF
SET HEADING OFF
```

After entering these commands, you use the SPOOL command as shown in the previous section to create the flat file.

The SET COLSEP command may be useful to delineate the columns. For more information, see the SET command in Chapter 7.

Sending Results to a Printer

To print query results, spool them to a file as described in the previous section. Then, instead of using SPOOL OFF, enter the command in the following form:

```
SPOOL OUT
```

SQL*Plus stops spooling and copies the contents of the spooled file to your host computer's standard (default) printer. SPOOL OUT does not delete the spool file after printing.

**Example 4–25
Sending Query Results
to a Printer**

To generate a final report and spool and print the results, create a command file named EMPRPT containing the following commands.

First, use EDIT to create the command file with your host operating system text editor. (Do not use INPUT and SAVE, or SQL*Plus will add a slash to the end of the file and will run the command file twice—once as a result of the semicolon and once due to the slash.)

```
SQL> EDIT EMPRPT
```

Next, enter the following commands into the file, using your text editor:

```
SPOOL TEMP
CLEAR COLUMNS
CLEAR BREAKS
CLEAR COMPUTES

COLUMN DEPTNO HEADING DEPARTMENT
COLUMN ENAME HEADING EMPLOYEE
COLUMN SAL HEADING SALARY FORMAT $99,999

BREAK ON DEPTNO SKIP 1 ON REPORT
```

```
COMPUTE SUM OF SAL ON DEPTNO
COMPUTE SUM OF SAL ON REPORT

SET PAGESIZE 21
SET NEWPAGE 0
SET LINESIZE 30

TTITLE CENTER 'A C M E   W I D G E T' SKIP 2 -
LEFT 'EMPLOYEE REPORT' RIGHT 'PAGE:' -
FORMAT 999 SQL.PNO SKIP 2

BTITLE CENTER 'COMPANY CONFIDENTIAL'

SELECT DEPTNO, ENAME, SAL
FROM EMP
ORDER BY DEPTNO;

SPOOL OUT
```

If you do not want to see the output on your screen, you can also add SET TERMOUT OFF to the beginning of the file and SET TERMOUT ON to the end of the file. Save the file (you automatically return to SQL*Plus). Now, run the command file EMPRPT:

```
SQL> @EMPRPT
```

SQL*Plus displays the output on your screen (unless you set TERMOUT to OFF), spools it to the file TEMP, and sends the contents of TEMP to your default printer:

```
          A C M E   W I D G E T

EMPLOYEE REPORT       PAGE:    1

DEPARTMENT EMPLOYEE     SALARY
---------- ---------- --------
        10 CLARK        $2,450
           KING         $5,000
           MILLER       $1,300
**********            --------
sum                     $8,750
```

```
           20 SMITH             $800
              ADAMS           $1,100
              FORD            $3,000
              SCOTT           $3,000
              JONES           $2,975

**********               --------
sum                       $10,875


            COMPANY CONFIDENTIAL


           A C M E   W I D G E T

EMPLOYEE REPORT      PAGE:   2

DEPARTMENT EMPLOYEE     SALARY
---------- ---------- --------
           30 ALLEN           $1,600
              BLAKE           $2,850
              MARTIN          $1,250
              JAMES             $900
              TURNER          $1,500
              WARD            $1,250
**********               --------
sum                        $9,400
**********               --------
sum                       $29,025


            COMPANY CONFIDENTIAL
```

5

Accessing SQL Databases

This chapter explains how to access databases through SQL*Plus, and discusses the following topics:

- connecting to the default database

- connecting to a remote database

- copying data from one database to another

- copying data between tables on one database

Read this chapter while sitting at your computer and try out the example shown. Before beginning, make sure you have access to the sample tables described in Chapter 1.

Connecting to the Default Database

In order to access data in a given database, you must first connect to the database. When you start SQL*Plus, you normally connect to your default Oracle database under the username and password you enter while starting. Once you have logged in, you can connect under a different username with the SQL*Plus CONNECT command. The username and password must be valid for the database.

For example, to connect the username TODD to the default database using the password FOX, you could enter

```
SQL> CONNECT TODD/FOX
```

If you omit the username and password, SQL*Plus prompts you for them. You also have the option of typing only the username following CONNECT and omitting the password (SQL*Plus then prompts for the password). Because CONNECT first disconnects you from your current database, you will be left unconnected to any database if you use an invalid username and password in your CONNECT command.

If you log on or connect as a user whose account has expired, SQL*Plus prompts you to change your password before you can connect.

If an account is locked, a message is displayed and connection as this user is not permitted until the account is unlocked by your DBA.

You can disconnect the username currently connected to Oracle without leaving SQL*Plus by entering the SQL*Plus command DISCONNECT at the SQL*Plus command prompt.

The default database is configured at an operating system level by setting operating system environment variables, symbols or, possibly, by editing an Oracle specific configuration file. Refer to your Oracle documentation for your operating system for more information.

Connecting to a Remote Database

Many large installations run Oracle on more than one computer. Such computers are often connected in a network, which permits programs on different computers to exchange data rapidly and efficiently. Networked computers can be physically near each other, or can be separated by large distances and connected by telecommunication links.

Databases on other computers or databases on your host computer other than your default database are called *remote databases.* You can access remote databases if the desired database has SQL*Net and both databases have compatible network drivers.

You can connect to a remote database in one of two ways:

- from within SQL*Plus, using the CONNECT command
- as you start SQL*Plus, using the SQLPLUS command

Connecting to a Remote Database from within SQL*Plus

To connect to a remote database using CONNECT, include a SQL*Net database specification in the CONNECT command in one of the following forms (the username and password you enter must be valid for the database to which you wish to connect):

- `CONNECT SCOTT@database_specification`
- `CONNECT SCOTT/TIGER@database_specification`

SQL*Plus prompts you for a password as needed, and connects you to the specified database. This database becomes the default database until you CONNECT again to another database, DISCONNECT, or leave SQL*Plus.

If you log on or connect as a user whose account has expired, SQL*Plus prompts you to change your password before you can connect.

If an account is locked, a message is displayed and connection as this user is not permitted until the account is unlocked by your DBA.

When you connect to a remote database in this manner, you can use the complete range of SQL and SQL*Plus commands and PL/SQL blocks on the database.

The exact string you enter for the database specification depends upon the SQL*Net protocol your computer uses. For more information, see CONNECT in Chapter 7 and the SQL*Net manual appropriate for your protocol, or contact your DBA.

Connecting to a Remote Database as You Start SQL*Plus

To connect to a remote database when you start SQL*Plus, include the SQL*Net database specification in your SQLPLUS command in one of the following forms:

- `SQLPLUS SCOTT@database_specification`
- `SQLPLUS SCOTT/TIGER@database_specification`

You must use a username and password valid for the remote database and substitute the appropriate database specification for the remote database. SQL*Plus prompts you for username and password as needed, starts SQL*Plus, and connects you to the specified database.

This database becomes the default database until you CONNECT to another database, DISCONNECT, or leave SQL*Plus.

If you log on or connect as a user whose account has expired, SQL*Plus prompts you to change your password before you can connect.

If an account is locked, a message is displayed and connection as this user is not permitted until the account is unlocked by your DBA.

Once again, you can manipulate tables in the remote database directly after you connect in this manner.

> **Note:** Do not confuse the @ symbol of the connect string with the @ command used to run a command file.

Copying Data from One Database to Another

Use the SQL*Plus COPY command to copy data between databases and between tables on the same database. With the COPY command, you can copy data between databases in the following ways:

- copy data from a remote database to your local database
- copy data from your local (default) database to a remote database (on most systems)
- copy data from one remote database to another remote database (on most systems)

> **Note:** In general, the COPY command was designed to be used for copying data between Oracle and non-Oracle databases. You should use SQL commands (CREATE TABLE AS and INSERT) to copy data between Oracle databases.

Understanding COPY Command Syntax

You enter the COPY command in the following form:

```
COPY FROM database TO database action –
  destination_table (column_name, column_name, –
  column_name ...) USING query
```

Here is a sample COPY command:

```
COPY FROM SCOTT/TIGER@BOSTONDB –
  TO TODD/FOX@CHICAGODB –
  CREATE NEWDEPT (DNUMBER, DNAME, CITY)–
  USING SELECT * FROM DEPT
```

To specify a database in the FROM or TO clause, you must have a valid username and password for the local and remote database(s) and know the appropriate database specification(s). COPY obeys Oracle security, so the username you specify must have been granted access to tables for you to have access to tables. For information on what databases are available to you, contact your DBA.

When you copy to your local database from a remote database, you can omit the TO clause. When you copy to a remote database from your local database, you can omit the FROM clause. When you copy between remote databases, you must include both clauses.

The COPY command behaves differently based on whether the destination table already exists and on the action clause you enter (CREATE in the example above). See "Controlling Treatment of the Destination Table" later in this chapter.

By default, the copied columns have the same names in the destination table that they have in the source table. If you want to give new names to the columns in the destination table, enter the new names in parentheses after the destination table name. If you enter any column names, you must enter a name for every column you are copying.

> **Note:** To enable the copying of data between Oracle and non-Oracle databases, NUMBER columns are changed to DECIMAL columns in the destination table. Hence, if you are copying between Oracle databases, a NUMBER column with no precision will be changed to a DECIMAL(38) column. When copying between Oracle databases, you should use SQL commands (CREATE TABLE AS and INSERT) or you should ensure that your columns have a precision specified.

The USING clause specifies a query that names the source table and specifies the data that COPY copies to the destination table. You can use any form of the SQL SELECT command to select the data that the COPY command copies.

Here is an example of a COPY command that copies only two columns from the source table, and copies only those rows in which the value of DEPTNO is 30:

```
SQL> COPY FROM SCOTT/TIGER@BOSTONDB —
>   REPLACE EMPCOPY2 —
>   USING SELECT ENAME, SAL —
>   FROM EMPCOPY —
>   WHERE DEPTNO = 30
```

You may find it easier to enter and edit long COPY commands in command files rather than trying to enter them directly at the command prompt.

Controlling Treatment of the Destination Table

You control the treatment of the destination table by entering one of four control clauses—REPLACE, CREATE, INSERT, or APPEND.

The REPLACE clause names the table to be created in the destination database and specifies the following actions:

- If the destination table already exists, COPY drops the existing table and replaces it with a table containing the copied data.

- If the destination table does not already exist, COPY creates it using the copied data.

You can use the CREATE clause to avoid accidentally writing over an existing table. CREATE specifies the following actions:

- If the destination table already exists, COPY reports an error and stops.

- If the destination table does not already exist, COPY creates the table using the copied data.

Use INSERT to insert data into an existing table. INSERT specifies the following actions:

- If the destination table already exists, COPY inserts the copied data in the destination table.

- If the destination table does not already exist, COPY reports an error and stops.

Use APPEND when you want to insert data in an existing table, or create a new table if the destination table does not exist. APPEND specifies the following actions:

- If the destination table already exists, COPY inserts the copied data in the destination table.

- If the table does not already exist, COPY creates the table and then inserts the copied data in it.

**Example 5–1
Copying from a
Remote Database to
Your Local Database
Using CREATE**

To copy EMP from a remote database into a table called EMPCOPY on your own database, enter the following command:

> **Note:** See your DBA for an appropriate username, password, and database specification for a remote computer that contains a copy of EMP.

```
SQL> COPY FROM SCOTT/TIGER@BOSTONDB –
>  CREATE EMPCOPY –
>  USING SELECT * FROM EMP
```

SQL*Plus displays the following messages:

```
Array fetch/bind size is 20. (arraysize is 20)
Will commit when done. (copycommit is 0)
Maximum long size is 80. (long is 80)
```

SQL*Plus then creates the table EMPCOPY, copies the rows, and displays the following additional messages:

```
Table EMPCOPY created.
 14 rows selected from SCOTT@BOSTONDB.
 14 rows inserted into EMPCOPY.
 14 rows committed into EMPCOPY at DEFAULT HOST connection.
```

In this COPY command, the FROM clause directs COPY to connect you to the database with the specification D:BOSTON–MFG as SCOTT, with the password TIGER.

Notice that you do not need a semicolon at the end of the command; COPY is a SQL*Plus command, not a SQL command, even though it contains a query. Since most COPY commands are longer than one line, you must use a hyphen (–), optionally preceded by a space, at the end of each line except the last.

**Interpreting the
Messages that COPY
Displays**

The first three messages displayed by COPY show the values of SET command variables that affect the COPY operation. The most important one is LONG, which limits the length of a LONG column's value. (LONG is a datatype, similar to CHAR.) If the source table contains a LONG column, COPY truncates values in that column to the length specified by the system variable LONG.

The variable ARRAYSIZE limits the number of rows that SQL*Plus fetches from the database at one time. This number of rows makes up a *batch*. The variable COPYCOMMIT sets the number of batches after which COPY commits changes to the database. (If you set COPYCOMMIT to zero, COPY commits changes only after all batches are copied.) For more information on the variables of the SET command,

including how to change their settings, see the SET command in Chapter 7.

After listing the three system variables and their values, COPY tells you if a table was dropped, created, or updated during the copy. Then COPY lists the number of rows selected, inserted, and committed.

Specifying Another User's Table

You can refer to another user's table in a COPY command by qualifying the table name with the username, just as you would in your local database, or in a query with a database link.

For example, to make a local copy of a table named DEPT, owned by the username ADAMS on the database associated with the SQL*Net connect string BOSTONDB, you would enter

```
SQL> COPY FROM SCOTT/TIGER@BOSTONDB -
>   CREATE EMPCOPY2 -
>   USING SELECT * FROM ADAMS.DEPT
```

Of course, you could get the same result by instructing COPY to log in to the remote database as ADAMS. You cannot do that, however, unless you know the password associated with the username ADAMS.

Copying Data between Tables on One Database

You can copy data from one table to another in a single database (local or remote). To copy between tables in your local database, specify your own username and password and the database specification for your local database in either a FROM or a TO clause (omit the other clause):

```
SQL> COPY FROM SCOTT/TIGER@MYDATABASE -
>   INSERT EMPCOPY2 -
>   USING SELECT * FROM EMP
```

To copy between tables on a remote database, include the same username, password, and database specification in the FROM and TO clauses:

```
SQL> COPY FROM SCOTT/TIGER@BOSTONDB -
>   TO SCOTT/TIGER@BOSTONDB -
>   INSERT EMPCOPY2 -
>   USING SELECT * FROM EMP
```

Reference

Starting SQL*Plus and Getting Help

This chapter explains how to access SQL*Plus from the operating system prompt, and discusses the following topics:

- starting SQL*Plus using the SQLPLUS command
- getting help

Starting SQL*Plus Using the SQLPLUS Command

You can start SQL*Plus from the operating system prompt by entering the SQLPLUS command in the following form:

```
SQLPLUS [[-S[ILENT]] [logon] [start]]|-|-?
```

where:

logon   Requires the following syntax:

```
username[/password]
[@database_specification]|/|/NOLOG
```

start   Allows you to enter the name of a command file
                    and arguments. SQL*Plus passes the arguments to
                    the command file as though you executed the file
                    using the SQL*Plus START command. The *start*
                    clause requires the following syntax:

```
@file_name[.ext][arg ...]
```

                    See the START command in Chapter 7 for more
                    information.

You have the option of entering *logon.* If you do not specify *logon* and do specify *start,* SQL*Plus assumes that the first line of the command file contains a valid logon. If neither *start* nor *logon* are specified, SQL*Plus prompts for logon information.

Refer to the following list for a description of each term or clause:

-S[ILENT]   Suppresses all SQL*Plus information and prompt
                    messages, including the command prompt, the
                    echoing of commands, and the banner normally
                    displayed when you start SQL*Plus. Use SILENT to
                    invoke SQL*Plus within another program so that
                    the use of SQL*Plus is invisible to the user.

username[/password]
                    Represent the username and password with which
                    you wish to start SQL*Plus and connect to Oracle. If
                    you omit *username* and *password,* SQL*Plus prompts
                    you for them. If you enter a slash (/) or simply
                    enter [Return] to the prompt for *username,* SQL*Plus
                    logs you in using a default logon (see "/" below).

                    If you omit only *password,* SQL*Plus prompts you
                    for *password.* When prompting, SQL*Plus does not
                    display *password* on your terminal screen.

database_specification
                Consists of a SQL*Net connection string. The exact syntax depends upon the SQL*Net communications protocol your Oracle installation uses. For more information, refer to the SQL*Net manual appropriate for your protocol or contact your DBA.

/   Represents a default logon using operating system authentication. You cannot enter a *database_specification* if you use a default logon. In a default logon, SQL*Plus typically attempts to log you in using the username OPS$*name*, where *name* is your operating system username. See the *Oracle8 Server Administrator's Guide* for information about operating system authentication.

/NOLOG   Establishes no initial connection to Oracle. Before issuing any SQL commands, you must issue a CONNECT command to establish a valid logon. Use /NOLOG when you want to have a SQL*Plus command file prompt for the username, password, or database specification. The first line of this command file is not assumed to contain a logon.

–   Displays the usage and syntax for the SQLPLUS command, and then returns control to the operating system.

–?   Displays the current version and level number for SQL*Plus, and then returns control to the operating system. Do not enter a space between the hyphen (–) and the question mark (?).

The SQL*Plus command may be known by a different name under some operating systems, for example, plus80. See your SQL*Plus installation documentation for further information on your operating system.

Setting Up the Site Profile

SQL*Plus supports a Site Profile, a SQL*Plus command file created by the database administrator. This file is generally named GLOGIN with an extension of SQL. SQL*Plus executes this command file whenever any user starts SQL*Plus and SQL*Plus establishes the Oracle connection. The Site Profile allows the DBA to set up SQL*Plus environment defaults for all users at a particular site; users cannot directly access the Site Profile. The default name and location of the Site Profile depend on your system. Site Profiles are described in more detail in the Oracle installation and user's manual(s) provided for your operating system.

Setting Up the User Profile

SQL*Plus also supports a User Profile, executed after the Site Profile. SQL*Plus searches for a file named LOGIN with the extension SQL in your current directory. If SQL*Plus does not find the file there, SQL*Plus will search a system-dependent path to find the file. Some operating systems may not support this path search.

Receiving a Return Code

If you fail to log in successfully to SQL*Plus because your username or password is invalid or some other error, SQL*Plus will return an error status equivalent to an EXIT FAILURE command. See the EXIT command in this chapter for further information.

Example 6–1 Starting SQL*Plus

To start SQL*Plus with *username* SCOTT and *password* TIGER, enter

```
SQLPLUS SCOTT/TIGER
```

To start SQL*Plus, as above, and to make POLICY the default database (where POLICY is a valid SQL*Net database connection string), enter

```
SQLPLUS SCOTT/TIGER@POLICY
```

To start SQL*Plus with *username* SCOTT and *password* TIGER and run a command file named STARTUP with the extension SQL, enter

```
SQLPLUS SCOTT/TIGER @STARTUP
```

Note the space between TIGER and @STARTUP.

Example 6–2 Displaying the SQLPLUS syntax

To display the syntax of the SQLPLUS command, enter

```
SQLPLUS -
```

SQL*Plus displays the following

```
Usage: SQLPLUS [<option>] [<user>[/password>] [@<host>]]
               [@<startfile> [<parm1>] [<parm2>] ...]
where <option> ::= {-s|-?}
-s for silent mode and -? to obtain version number
```

Getting Help

To access online help for SQL*Plus commands, you can type HELP followed by the command name at the SQL command prompt. For example:

```
SQL> HELP ACCEPT
```

If you get a response that help is unavailable, consult your database administrator. See the HELP command in Chapter 7 for more information.

7

Command Reference

This chapter contains descriptions of SQL*Plus commands, listed alphabetically. Use this chapter for reference only. Each description contains the following parts:

Purpose   Discusses the basic use(s) of the command.

Syntax   Shows how to enter the command. Refer to Chapter 1 for an explanation of the syntax notation.

Terms and Clauses   Describes the function of each term or clause appearing in the syntax.

Usage Notes   Provides additional information on how the command works and on uses of the command.

Examples   Gives one or more examples of the command.

A summary table that lists and briefly describes SQL*Plus commands precedes the individual command descriptions.

You can continue a long SQL*Plus command by typing a hyphen at the end of the line and pressing [Return]. If you wish, you can type a space before typing the hyphen. SQL*Plus displays a right angle-bracket (>) as a prompt for each additional line.

You do not need to end a SQL*Plus command with a semicolon. When you finish entering the command, you can just press [Return]. If you wish, however, you can enter a semicolon at the end of a SQL*Plus command.

SQL*Plus Command Summary

Command	Description
@	Runs the specified command file.
@@	Runs a command file.
/	Executes the SQL command or PL/SQL block.
ACCEPT	Reads a line of input and stores it in a given user variable.
APPEND	Adds specified text to the end of the current line in the buffer.
ATTRIBUTE	Specifies display characteristics for a given attribute of an Object Type column, and lists the current display characteristics for a single attribute or all attributes.
BREAK	Specifies where and how formatting will change in a report, or lists the current break definition.
BTITLE	Places and formats a specified title at the bottom of each report page, or lists the current BTITLE definition.
CHANGE	Changes text on the current line in the buffer.
CLEAR	Resets or erases the current value or setting for the specified option, such as BREAKS or COLUMNS.
COLUMN	Specifies display characteristics for a given column, or lists the current display characteristics for a single column or for all columns.
COMPUTE	Calculates and prints summary lines, using various standard computations, on subsets of selected rows, or lists all COMPUTE definitions.
CONNECT	Connects a given username to Oracle.
COPY	Copies data from a query to a table in a local or remote database.
DEFINE	Specifies a user variable and assigns it a CHAR value, or lists the value and variable type of a single variable or all variables.
DEL	Deletes one or more lines of the buffer.
DESCRIBE	Lists the column definitions for the specified table, view, or synonym or the specifications for the specified function or procedure.

Command	Description
DISCONNECT	Commits pending changes to the database and logs the current username off Oracle, but does not exit SQL*Plus.
EDIT	Invokes a host operating system text editor on the contents of the specified file or on the contents of the buffer.
EXECUTE	Executes a single PL/SQL statement.
EXIT	Terminates SQL*Plus and returns control to the operating system.
GET	Loads a host operating system file into the SQL buffer.
HELP	Accesses the SQL*Plus help system.
HOST	Executes a host operating system command without leaving SQL*Plus.
INPUT	Adds one or more new lines after the current line in the buffer.
LIST	Lists one or more lines of the SQL buffer.
PASSWORD	Allows a password to be changed without echoing the password on an input device.
PAUSE	Displays an empty line followed by a line containing text, then waits for the user to press [Return], or displays two empty lines and waits for the user's response.
PRINT	Displays the current value of a bind variable.
PROMPT	Sends the specified message or a blank line to the user's screen.
REMARK	Begins a comment in a command file.
REPFOOTER	Places and formats a specified report footer at the bottom of each report, or lists the current REPFOOTER definition.
REPHEADER	Places and formats a specified report header at the top of each report, or lists the current REPHEADER definition.
RUN	Lists and executes the SQL command or PL/SQL block currently stored in the SQL buffer.
SAVE	Saves the contents of the SQL buffer in a host operating system file (a command file).

Command	Description
SET	Sets a system variable to alter the SQL*Plus environment for your current session.
SHOW	Shows the value of a SQL*Plus system variable or the current SQL*Plus environment.
SPOOL	Stores query results in an operating system file and, optionally, sends the file to a printer.
START	Executes the contents of the specified command file.
STORE	Saves attributes of the current SQL*Plus environment in a host operating system file (a command file).
TIMING	Records timing data for an elapsed period of time, lists the current timer's title and timing data, or lists the number of active timers.
TTITLE	Places and formats a specified title at the top of each report page, or lists the current TTITLE definition.
UNDEFINE	Deletes one or more user variables that you defined either explicitly (with the DEFINE command) or implicitly (with an argument to the START command).
VARIABLE	Declares a bind variable that can be referenced in PL/SQL.
WHENEVER OSERROR	Exits SQL*Plus if an operating system command generates an error.
WHENEVER SQLERROR	Exits SQL*Plus if a SQL command or PL/SQL block generates an error.

@ ("at" sign)

Purpose   Runs the specified command file.

Syntax   `@ file_name[.ext] [arg...]`

Terms and Clauses   Refer to the following list for a description of each term or clause:

`file_name[.ext]`   Represents the command file you wish to run. If you omit *ext*, SQL*Plus assumes the default command-file extension (normally SQL). For information on changing the default extension, see the SUFFIX variable of the SET command in this chapter.

When you enter @ *file_name.ext*, SQL*Plus searches for a file with the filename and extension you specify in the current default directory. If SQL*Plus does not find such a file, SQL*Plus will search a system-dependent path to find the file. Some operating systems may not support the path search. Consult the Oracle installation and user's manual(s) provided for your operating system for specific information related to your operating system environment.

`arg...`   Represent data items you wish to pass to parameters in the command file. If you enter one or more arguments, SQL*Plus substitutes the values into the parameters (&1, &2, and so forth) in the command file. The first argument replaces each occurrence of &1, the second replaces each occurrence of &2, and so forth.

The @ command DEFINEs the parameters with the values of the arguments; if you run the command file again in this session, you can enter new arguments or omit the arguments to use the current values.

For more information on using parameters, refer to the subsection "Passing Parameters through the START Command" under "Writing Interactive Commands" in Chapter 3.

Usage Notes   You can include in a command file any command you would normally enter interactively (typically, SQL, SQL*Plus commands, or PL/SQL blocks).

An EXIT or QUIT command used in a command file terminates SQL*Plus.

The @ command functions similarly to START.

If the START command is disabled (see "Disabling SQL*Plus, SQL, and PL/SQL Commands" in Appendix E), this will also disable the @ command. See START in this chapter for information on the START command.

SQL*Plus removes the SQLTERMINATOR (a semicolon by default) before the @ command is issued. A workaround for this is to add another SQLTERMINATOR. See the SQLTERMINATOR variable of the SET command in this chapter for more information.

Example   To run a command file named PRINTRPT with the extension SQL, enter

```
SQL> @PRINTRPT
```

To run a command file named WKRPT with the extension QRY, enter

```
SQL> @WKRPT.QRY
```

@@ (double "at" sign)

Purpose   Runs a command file. This command is identical to the @ ("at" sign) command except that it looks for the specified command file in the same path as the command file from which it was called.

Syntax   `@@ file_name[.ext]`

Terms and Clauses   Refer to the following for a description of the term or clause:

`file_name[.ext]`   Represents the nested command file you wish to run. If you omit *ext*, SQL*Plus assumes the default command-file extension (normally SQL). For information on changing the default extension, see the SUFFIX variable of the SET command in this chapter.

When you enter *@@file_name.ext* from within a command file, SQL*Plus runs *file_name.ext* from the same directory as the command file. When you enter *@@file_name.ext* interactively, SQL*Plus runs *file_name.ext* from the current working directory. If SQL*Plus does not find such a file, SQL*Plus searches a system-dependent path to find the file. Some operating systems may not support the path search. Consult the Oracle installation and user's manual(s) provided for your operating system for specific information related to your operating system environment.

Usage Notes   You can include in a command file any command you would normally enter interactively (typically, SQL or SQL*Plus commands).

An EXIT or QUIT command used in a command file terminates SQL*Plus.

The @@ command functions similarly to START.

If the START command is disabled, this will also disable the @@ command. See START in this chapter for further information on the START command.

SQL*Plus removes the SQLTERMINATOR (a semicolon by default) before the @@ command is issued. A workaround for this is to add another SQLTERMINATOR. See the SQLTERMINATOR variable of the SET command in this chapter for more information.

Example   Suppose that you have the following command file named PRINTRPT:

```
SELECT * FROM EMP
@EMPRPT
@@ WKRPT
```

When you START PRINTRPT and it reaches the @ command, it looks for the command file named EMPRPT in the current working directory and runs it. When PRINTRPT reaches the @@ command, it looks for the command file named WKRPT in the same path as PRINTRPT and runs it.

/ (slash)

Purpose   Executes the SQL command or PL/SQL block currently stored in the SQL buffer.

Syntax   /

Usage Notes   You can enter a slash (/) at the command prompt or at a line number prompt of a multi-line command.

The slash command functions similarly to RUN, but does not list the command in the buffer on your screen.

Executing a SQL command or PL/SQL block using the slash command will not cause the current line number in the SQL buffer to change unless the command in the buffer contains an error. In that case, SQL*Plus changes the current line number to the number of the line containing the error.

Example   To see the SQL command(s) you will execute, you can list the contents of the buffer:

```
SQL> LIST
  1* SELECT ENAME, JOB FROM EMP WHERE ENAME = 'JAMES'
```

Enter a slash (/) at the command prompt to execute the command in the buffer:

```
SQL> /
```

For the above query, SQL*Plus displays the following output:

```
ENAME      JOB
---------- ---------
JAMES      CLERK
```

ACCEPT

Purpose   Reads a line of input and stores it in a given user variable.

Syntax

```
ACC[EPT] variable [NUM[BER]|CHAR|DATE] [FOR[MAT] format]
    [DEF[AULT] default] [PROMPT text|NOPR[OMPT]] [HIDE]
```

Terms and Clauses   Refer to the following list for a description of each term or clause:

variable   Represents the name of the variable in which you wish to store a value. If *variable* does not exist, SQL*Plus creates it.

NUM[BER]   Makes the datatype of *variable* the datatype NUMBER. If the reply does not match the datatype, ACCEPT gives an error message and prompts again.

CHAR   Makes the datatype of *variable* the datatype CHAR. The maximum CHAR length limit is 240 bytes. If a multi-byte character set is used, one CHAR may be more than one byte in size.

DATE   Makes reply a valid DATE format. If the reply is not a valid DATE format, ACCEPT gives an error message and prompts again. The datatype is CHAR.

FOR[MAT]   Specifies the input format for the reply. If the reply does not match the specified format, ACCEPT gives an error message and prompts again for a reply. The format element must be a text constant such as A10 or 9.999. See the COLUMN command in this chapter for a complete list of format elements.

    Oracle date formats such as 'dd/mm/yy' are valid when the datatype is DATE. DATE without a specified format defaults to the Oracle NLS_DATE_FORMAT of the current session. See the *Oracle8 Server Administrator's Guide* and the *Oracle8 Server SQL Reference Guide* for information on Oracle date formats.

DEF[AULT]   Sets the default value if a reply is not given. The reply must be in the specified format if defined.

PROMPT *text*   Displays *text* on-screen before accepting the value of *variable* from the user.

NOPR[OMPT]	Skips a line and waits for input without displaying a prompt.
HIDE	Suppresses the display as you type the reply.

To display or reference variables, use the DEFINE command. See the DEFINE command in this chapter for more information.

Examples To display the prompt "Password: ", place the reply in a CHAR variable named PSWD, and suppress the display, enter

```
SQL> ACCEPT pswd CHAR PROMPT 'Password:  ' HIDE
```

To display the prompt "Enter weekly salary: " and place the reply in a NUMBER variable named SALARY with a default of 000.0, enter

```
SQL> ACCEPT salary NUMBER FORMAT '999.99' DEFAULT '000.0' –
>       PROMPT 'Enter weekly salary:  '
```

To display the prompt "Enter date hired: " and place the reply in a DATE variable named HIRED with the format "dd/mm/yy" and a default of "01/01/94", enter

```
SQL> ACCEPT hired DATE FORMAT 'dd/mm/yy' DEFAULT '01/01/94'–
>       PROMPT 'Enter date hired:  '
```

To display the prompt "Enter employee lastname: " and place the reply in a CHAR variable named LASTNAME, enter

```
SQL> ACCEPT lastname CHAR FORMAT 'A20' –
>       PROMPT 'Enter employee lastname:  '
```

APPEND

Purpose   Adds specified text to the end of the current line in the SQL buffer.

Syntax   `A[PPEND] text`

Terms and Clauses   Refer to the following for a description of the term or clause:

     `text`   Represents the text you wish to append. If you wish to separate *text* from the preceding characters with a space, enter two spaces between APPEND and *text.*

           To APPEND *text* that ends with a semicolon, end the command with two semicolons (SQL*Plus interprets a single semicolon as an optional command terminator).

Examples   To append a space and the column name DEPT to the second line of the buffer, make that line the current line by listing the line as follows:

```
SQL> 2
  2* FROM EMP,
```

Now enter APPEND:

```
SQL> APPEND  DEPT
SQL> 2
  2* FROM EMP, DEPT
```

Notice the double space between APPEND and DEPT. The first space separates APPEND from the characters to be appended; the second space becomes the first appended character.

To append a semicolon to the line, enter

```
SQL> APPEND ;;
```

SQL*Plus appends the first semicolon to the line and interprets the second as the terminator for the APPEND command.

ATTRIBUTE

Purpose
: Specifies display characteristics for a given attribute of an Object Type column, such as format for NUMBER data.

: Also lists the current display characteristics for a single attribute or all attributes.

Syntax
: ```
ATTRIBUTE [type_name.attribute_name [option ...]]
```

: where *option* represents one of the following clauses:

: ```
ALI[AS] alias
CLE[AR]
FOR[MAT] format
LIKE {type_name.attribute_name|alias}
ON|OFF
```

Terms and Clauses
: Enter ATTRIBUTE followed by *type_name.attribute_name* and no other clauses to list the current display characteristics for only the specified attribute. Enter ATTRIBUTE with no clauses to list all current attribute display characteristics.

: Refer to the following list for a description of each term or clause:

: *type_name.attribute_name*

: > Identifies the data item (typically the name of an attribute) within the set of attributes for a given object of Object Type, *type_name*.

: > If you select objects of the same Object Type, an ATTRIBUTE command for that *type_name.attribute_name* will apply to all such objects you reference in that session.

: `ALI[AS]` *alias*
: > Assigns a specified alias to a *type_name.attribute_name*, which can be used to refer to the *type_name.attribute_name* in other ATTRIBUTE commands.

: `CLE[AR]`
: > Resets the display characteristics for the *attribute_name*. The format specification must be a text constant such as A10 or $9,999—not a variable.

: `FOR[MAT]` *format*
: > Specifies the display format of the column. The format specification must be a text constant such as A10 or $9,999—not a variable.

```
LIKE {type_name.attribute_name|alias}
```
                    Copies the display characteristics of another attribute. LIKE copies only characteristics not defined by another clause in the current ATTRIBUTE command.

<u>ON</u>|OFF   Controls the status of display characteristics for a column. OFF disables the characteristics for an attribute without affecting the characteristics' definition. ON reinstates the characteristics.

Usage Notes   You can enter any number of ATTRIBUTE commands for one or more attributes. All attribute characteristics set for each attribute remain in effect for the remainder of the session, until you turn the attribute OFF, or until you use the CLEAR COLUMN command. Thus, the ATTRIBUTE commands you enter can control an attribute's display characteristics for multiple SQL SELECT commands.

When you enter multiple ATTRIBUTE commands for the same attribute, SQL*Plus applies their clauses collectively. If several ATTRIBUTE commands apply the same clause to the same attribute, the last one entered will control the output.

Examples   To make the ENAME attribute of the Object Type EMP_TYPE 20 characters wide, enter

```
SQL> ATTRIBUTE EMP_TYPE.ENAME FORMAT A20
```

To format the SAL attribute of the Object Type EMP_TYPE so that it shows millions of dollars, rounds to cents, uses commas to separate thousands, and displays $0.00 when a value is zero, enter

```
SQL> ATTRIBUTE EMP_TYPE.SAL FORMAT $9,999,990.99
```

BREAK

Purpose
Specifies where and how formatting will change in a report, such as

- suppressing display of duplicate values for a given column
- skipping a line each time a given column value changes
- printing COMPUTEd figures each time a given column value changes or at the end of the report (see also the COMPUTE command)

Also lists the current BREAK definition.

Syntax
```
BRE[AK] [ON report_element [action [action]]] ...
```
where:

`report_element`	Requires the following syntax:
	`{column|expr|ROW|REPORT}`
`action`	Requires the following syntax:
	`[SKI[P] n|[SKI[P]] PAGE]` `[NODUP[LICATES]|DUP[LICATES]]`

Terms and Clauses
Refer to the following list for a description of each term or clause:

`ON column [action [action]]`

When you include action(s), specifies action(s) for SQL*Plus to take whenever a break occurs in the specified column (called the *break column*). (*column* cannot have a table or view appended to it. To achieve this, you can alias the column in the SQL statement.) A break is one of three events:

- a change in the value of a column or expression
- the output of a row
- the end of a report

When you omit action(s), BREAK ON *column* suppresses printing of duplicate values in *column* and marks a place in the report where SQL*Plus will perform the computation you specify in a corresponding COMPUTE command.

You can specify ON *column* one or more times. If you specify multiple ON clauses, as in

```
SQL> BREAK ON DEPTNO SKIP PAGE ON JOB -
>  SKIP 1 ON SAL SKIP 1
```

the first ON clause represents the *outermost break* (in this case, ON DEPTNO) and the last ON clause represents the *innermost break* (in this case, ON SAL). SQL*Plus searches each row of output for the specified break(s), starting with the outermost break and proceeding—in the order you enter the clauses—to the innermost. In the example, SQL*Plus searches for a change in the value of DEPTNO, then JOB, then SAL.

Next, SQL*Plus executes actions beginning with the action specified for the innermost break and proceeding in reverse order toward the outermost break (in this case, from SKIP 1 for ON SAL toward SKIP PAGE for ON DEPTNO). SQL*Plus executes each action up to and including the action specified for the first occurring break encountered in the initial search.

If, for example, in a given row the value of JOB changes—but the values of DEPTNO and SAL remain the same—SQL*Plus skips *two* lines before printing the row (one as a result of SKIP 1 in the ON SAL clause and one as a result of SKIP 1 in the ON JOB clause).

Whenever you use ON *column*, you should also use an ORDER BY clause in the SQL SELECT command. Typically, the columns used in the BREAK command should appear in the same order in the ORDER BY clause (although all columns specified in the ORDER BY clause need not appear in the BREAK command). This prevents breaks from occurring at meaningless points in the report. The following SELECT command produces meaningful results:

```
SQL> SELECT DEPTNO, JOB, SAL, ENAME
  2   FROM EMP
  3   ORDER BY DEPTNO, JOB, SAL, ENAME;
```

All rows with the same DEPTNO print together on one page, and within that page all rows with the same JOB print in groups. Within each group of jobs, those jobs with the same SAL print in groups. Breaks in ENAME cause no action because ENAME does not appear in the BREAK command.

ON *expr* [*action* [*action*]]

When you include action(s), specifies action(s) for SQL*Plus to take when the value of the expression changes.

When you omit action(s), BREAK ON *expr* suppresses printing of duplicate values of *expr* and marks a place in the report where SQL*Plus will perform the computation you specify in a corresponding COMPUTE command.

You can use an expression involving one or more table columns or an alias assigned to a report column in a SQL SELECT or SQL*Plus COLUMN command. If you use an expression in a BREAK command, you must enter *expr* exactly as it appears in the SELECT command. If the expression in the SELECT command is a+b, for example, you cannot use b+a or (a+b) in a BREAK command to refer to the expression in the SELECT command.

The information given above for ON *column* also applies to ON *expr*.

ON ROW [*action* [*action*]]

When you include action(s), specifies action(s) for SQL*Plus to take when a SQL SELECT command returns a row. The ROW break becomes the innermost break regardless of where you specify it in the BREAK command. You should always specify an action when you BREAK on a row.

`ON REPORT [`*`action`*`]`

> Marks a place in the report where SQL*Plus will perform the computation you specify in a corresponding COMPUTE command. Use BREAK ON REPORT in conjunction with COMPUTE to print grand totals or other "grand" computed values.
>
> The REPORT break becomes the outermost break regardless of where you specify it in the BREAK command.
>
> Note that SQL*Plus will not skip a page at the end of a report, so you cannot use BREAK ON REPORT SKIP PAGE.

Refer to the following list for a description of each action:

`SKI[P]` *`n`*   Skips *n* lines before printing the row where the break occurred.

`[SKI[P]] PAGE`   Skips the number of lines that are defined to be a page before printing the row where the break occurred. The number of lines per page can be set via the PAGESIZE clause of the SET command. Note that PAGESIZE only changes the number of lines that SQL*Plus considers to be a page. Therefore, SKIP PAGE may not always cause a physical page break, unless you have also specified NEWPAGE 0. Note also that if there is a break after the last row of data to be printed in a report, SQL*Plus will not skip the page.

`NODUP[LICATES]`   Prints blanks rather than the value of a break column when the value is a duplicate of the column's value in the preceding row.

`DUP[LICATES]`   Prints the value of a break column in every selected row.

Enter BREAK with no clauses to list the current break definition.

Usage Notes   Each new BREAK command you enter replaces the preceding one.

To remove the BREAK command, use CLEAR BREAKS.

Example   To produce a report that prints duplicate job values, prints the average of SAL and inserts one blank line when the value of JOB changes, and *additionally* prints the sum of SAL and inserts another blank line when the value of DEPTNO changes, you could enter the following commands. (The example selects departments 10 and 30 and the jobs of clerk and salesman only.)

```
SQL> BREAK ON DEPTNO SKIP 1 ON JOB SKIP 1 DUPLICATES
SQL> COMPUTE SUM OF SAL ON DEPTNO
SQL> COMPUTE AVG OF SAL ON JOB
SQL> SELECT DEPTNO, JOB, ENAME, SAL FROM EMP
  2  WHERE JOB IN ('CLERK', 'SALESMAN')
  3  AND DEPTNO IN (10, 30)
  4  ORDER BY DEPTNO, JOB;
```

The following output results:

```
DEPTNO    JOB       ENAME            SAL
--------- --------- ---------- ---------
       10 CLERK     MILLER          1300
          ********             ---------
          avg                       1300

********                      ----------
sum                               1300

       30 CLERK     JAMES           1045
          ********             ----------
          avg                       1045

          SALESMAN  ALLEN           1760
          SALESMAN  MARTIN          1375
          SALESMAN  TURNER          1650
          SALESMAN  WARD            1375
          ********             ----------
          avg                       1540

********                      ----------
sum                               7205
```

BTITLE

Purpose	Places and formats a specified title at the bottom of each report page or lists the current BTITLE definition.
	For a description of the old form of BTITLE, see Appendix F.
Syntax	`BTI[TLE] [`*printspec* `[`*text*`|`*variable*`] ...]|[`<u>`OFF`</u>`|ON]`
Terms and Clauses	Refer to the TTITLE command for additional information on terms and clauses in the BTITLE command syntax.
	Enter BTITLE with no clauses to list the current BTITLE definition.
Usage Notes	If you do not enter a *printspec* clause before the first occurrence of *text*, BTITLE left justifies the text. SQL*Plus interprets BTITLE in the new form if a valid *printspec* clause (LEFT, SKIP, COL, and so on) immediately follows the command name.
Examples	To set a bottom title with CORPORATE PLANNING DEPARTMENT on the left and a date on the right, enter

```
SQL> BTITLE LEFT 'CORPORATE PLANNING DEPARTMENT' –
> RIGHT '27 Jun 1997'
```

To set a bottom title with CONFIDENTIAL in column 50, followed by six spaces and a date, enter

```
SQL> BTITLE COL 50 'CONFIDENTIAL' TAB 6 '27 Jun 1997'
```

CHANGE

Purpose Changes the first occurrence of text on the current line in the buffer.

Syntax C[HANGE] *sepchar old* [*sepchar* [*new* [*sepchar*]]]

Terms and Clauses Refer to the following list for a description of each term or clause:

sepchar Represents any non-alphanumeric character such as "/" or "!". Use a *sepchar* that does not appear in *old* or *new.* You can omit the space between CHANGE and the first *sepchar.*

old Represents the text you wish to change. CHANGE ignores case in searching for *old.* For example,

CHANGE /aq/aw

will find the first occurrence of "aq", "AQ", "aQ", or "Aq" and change it to "aw". SQL*Plus inserts the *new* text exactly as you specify it.

If *old* is prefixed with "...", it matches everything up to and including the first occurrence of *old.* If it is suffixed with "...", it matches the first occurrence of *old* and everything that follows on that line. If it contains an embedded "...", it matches everything from the preceding part of *old* through the following part of *old.*

new Represents the text with which you wish to replace *old.* If you omit new and, optionally, the second and third *sepchars,* CHANGE deletes *old* from the current line of the buffer.

Usage Notes CHANGE changes the first occurrence of the existing specified text on the current line of the buffer to the new specified text. The current line is marked with an asterisk (*) in the LIST output.

You can also use CHANGE to modify a line in the buffer that has generated an Oracle error. SQL*Plus sets the buffer's current line to the line containing the error so that you can make modifications.

To re-enter an entire line, you can type the line number followed by the new contents of the line. If you specify a line number larger than the number of lines in the buffer and follow the number with text, SQL*Plus adds the text in a new line at the end of the buffer. If you specify zero ("0") for the line number and follow the zero with text, then SQL*Plus inserts the line at the beginning of the buffer (that line becomes line 1).

Examples   Assume the current line of the buffer contains the following text:

```
4* WHERE JOB IS IN ('CLERK','SECRETARY','RECEPTIONIST')
```

Enter the following command:

```
SQL> C /RECEPTIONIST/GUARD/
```

The text in the buffer changes as follows:

```
4* WHERE JOB IS IN ('CLERK','SECRETARY','GUARD')
```

Or enter the following command:

```
SQL> C /'CLERK',.../'CLERK')/
```

The original line changes to

```
4* WHERE JOB IS IN ('CLERK')
```

Or enter the following command:

```
SQL> C /(...)/('COOK','BUTLER')/
```

The original line changes to

```
4* WHERE JOB IS IN ('COOK','BUTLER')
```

You can replace the contents of an entire line using the line number. This entry

```
SQL> 2  FROM EMP e1
```

causes the second line of the buffer to be replaced with

```
FROM EMP e1
```

Note that entering a line number followed by a string will replace the line regardless of what text follows the line number. Thus,

```
SQL> 2  c/old/new/
```

will change the second line of the buffer to be

```
2* c/old/new/
```

CLEAR

Purpose   Resets or erases the current value or setting for the specified option.

Syntax   `CL[EAR]` *option* `...`

where *option* represents one of the following clauses:

```
BRE[AKS]
BUFF[ER]
COL[UMNS]
COMP[UTES]
SCR[EEN]
SQL
TIMI[NG]
```

Terms and Clauses   Refer to the following list for a description of each term or clause:

`BRE[AKS]`	Removes the break definition set by the BREAK command.
`BUFF[ER]`	Clears text from the buffer. CLEAR BUFFER has the same effect as CLEAR SQL, unless you are using multiple buffers (see the SET BUFFER command in Appendix F).
`COL[UMNS]`	Resets column display attributes set by the COLUMN command to default settings for all columns. To reset display attributes for a single column, use the CLEAR clause of the COLUMN command. CLEAR COLUMNS also clears the ATTRIBUTEs for that column.
`COMP[UTES]`	Removes all COMPUTE definitions set by the COMPUTE command.
`SCR[EEN]`	Clears your screen.
`SQL`	Clears the text from SQL buffer. CLEAR SQL has the same effect as CLEAR BUFFER, unless you are using multiple buffers (see the SET BUFFER command in Appendix F).
`TIMI[NG]`	Deletes all timers created by the TIMING command.

Example   To clear breaks, enter

```
SQL> CLEAR BREAKS
```

To clear column definitions, enter

```
SQL> CLEAR COLUMNS
```

COLUMN

Purpose   Specifies display attributes for a given column, such as

- text for the column heading
- alignment of the column heading
- format for NUMBER data
- wrapping of column data

Also lists the current display attributes for a single column or all columns.

Syntax   COL[UMN] [{*column*|*expr*} [*option* ...]]

where *option* represents one of the following clauses:

```
ALI[AS] alias
CLE[AR]
FOLD_A[FTER]
FOLD_B[EFORE]
FOR[MAT] format
HEA[DING] text
JUS[TIFY] {L[EFT]|C[ENTER]|C[ENTRE]|R[IGHT]}
LIKE {expr|alias}
NEWL[INE]
NEW_V[ALUE] variable
NOPRI[NT]|PRI[NT]
NUL[L] text
OLD_V[ALUE] variable
ON|OFF
WRA[PPED]|WOR[D_WRAPPED]|TRU[NCATED]
```

Terms and Clauses   Enter COLUMN followed by *column* or *expr* and no other clauses to list the current display attributes for only the specified column or expression. Enter COLUMN with no clauses to list all current column display attributes.

Refer to the following list for a description of each term or clause:

{*column*|*expr*}   Identifies the data item (typically, the name of a column) in a SQL SELECT command to which the column command refers. If you use an expression in a COLUMN command, you must enter *expr* exactly as it appears in the SELECT command. If the expression in the SELECT command is a+b, for example, you cannot use b+a or (a+b) in a

COLUMN command to refer to the expression in the SELECT command.

If you select columns with the same name from different tables, a COLUMN command for that column name will apply to both columns. That is, a COLUMN command for the column ENAME applies to all columns named ENAME that you reference in this session. COLUMN ignores table name prefixes in SELECT commands. Also, spaces are ignored unless the name is placed in double quotes.

To format the columns differently, assign a unique alias to each column within the SELECT command itself (do not use the ALIAS clause of the COLUMN command) and enter a COLUMN command for each column's alias.

ALI[AS] *alias* Assigns a specified *alias* to a column, which can be used to refer to the column in BREAK, COMPUTE, and other COLUMN commands.

Note: A SQL*Plus *alias* is different from a SQL *alias*. See the *Oracle8 Server SQL Reference Manual* for further information on the SQL *alias*.

CLE[AR] Resets the display attributes for the column to default values.

To reset the attributes for all columns, use the CLEAR COLUMNS command. CLEAR COLUMNS also clears the ATTRIBUTEs for that column.

FOLD_A[FTER] Inserts a carriage return after the column heading and after each row in the column. SQL*Plus does not insert an extra carriage return after the last column in the SELECT list.

FOLD_B[EFORE] Inserts a carriage return before the column heading and before each row of the column. SQL*Plus does not insert an extra carriage return before the first column in the SELECT list.

FOR[MAT] *format* Specifies the display format of the column. The format specification must be a text constant such as A10 or $9,999—not a variable.

Character Columns   The default width of CHAR, NCHAR, VARCHAR2 (VARCHAR) and NVARCHAR2 (NCHAR VARYING) columns is the width of the column in the database. SQL*Plus formats these datatypes left-justified. If a value does not fit within the column width, SQL*Plus wraps or truncates the character string depending on the setting of SET WRAP.

A LONG, CLOB or NCLOB column's width defaults to the value of SET LONGCHUNKSIZE or SET LONG, whichever one is smaller.

A Trusted Oracle column of datatype MLSLABEL defaults to the width defined for the column in the database or the length of the column's heading, whichever is longer. The default display width for a Trusted Oracle ROWLABEL column is 15.

To change the width of a datatype or Trusted Oracle column to *n*, use FORMAT A*n*. (A stands for alphanumeric.) If you specify a width shorter than the column heading, SQL*Plus truncates the heading. If you specify a width for a LONG, CLOB, or NCLOB column, SQL*Plus uses the LONGCHUNKSIZE or the specified width, whichever is smaller, as the column width.

DATE Columns   The default width and format of unformatted DATE columns in SQL*Plus is derived from the NLS parameters in effect. Otherwise, the default width is A9. In Oracle8, the NLS parameters may be set in your database parameter file or may be environment variables or an equivalent platform-specific mechanism. They may also be specified for each session with the ALTER SESSION command. (See the documentation for the Oracle8 Server for a complete description of the NLS parameters).

You can change the format of any DATE column using the SQL function TO_CHAR in your SQL SELECT statement. You may also wish to use an explicit COLUMN FORMAT command to adjust the column width.

When you use SQL functions like TO_CHAR, Oracle automatically allows for a very wide column.

To change the width of a DATE column to *n*, use the COLUMN command with FORMAT A*n*. If you specify a width shorter than the column heading, the heading is truncated.

NUMBER Columns   To change a NUMBER column's width, use FORMAT followed by an element as specified in Table 7 – 1.

Element	*Example(s)*	*Description*
9	9999	Number of "9"s specifies number of significant digits returned. Blanks are displayed for leading zeroes. A zero (0) is displayed for a value of zero.
0	0999 9990	Displays a leading zero or a value of zero in this position as a 0.
$	$9999	Prefixes value with dollar sign.
B	B9999	Displays a zero value as blank, regardless of "0"s in the format model.
MI	9999MI	Displays "–" after a negative value. For a positive value, a trailing space is displayed.
S	S9999	Returns "+" for positive values and "–" for negative values in this position.
PR	9999PR	Displays a negative value in <angle brackets>. For a positive value, a leading and trailing space is displayed.
D	99D99	Displays the decimal character in this position, separating the integral and fractional parts of a number.
G	9G999	Displays the group separator in this position.
C	C999	Displays the ISO currency symbol in this position.
L	L999	Displays the local currency symbol in this position.
, (comma)	9,999	Displays a comma in this position.
. (period)	99.99	Displays a period (decimal point) in this position, separating the integral and fractional parts of a number.
V	999V99	Multiplies value by 10^n, where *n* is the number of "9"s after the "V".

Table 7 – 1 Number Formats

Element	Example(s)	Description
EEEE	9.999EEEE	Displays value in scientific notation (format must contain exactly four "E"s).
RN or **rn**	RN	Displays upper- or lowercase Roman numerals. Value can be an integer between 1 and 3999.
DATE	DATE	Displays value as a date in MM/DD/YY format; used to format NUMBER columns that represent Julian dates.

Table 7 – 1 Number Formats

The MI and PR format elements can only appear in the last position of a number format model. The S format element can only appear in the first or last position.

If a number format model does not contain the MI, S or PR format elements, negative return values automatically contain a leading negative sign and positive values automatically contain a leading space.

A number format model can contain only a single decimal character (D) or period (.), but it can contain multiple group separators (G) or commas (,). A group separator or comma cannot appear to the right of a decimal character or period in a number format model.

SQL*Plus formats NUMBER data right-justified. A NUMBER column's width equals the width of the heading or the width of the FORMAT plus one space for the sign, whichever is greater. If you do not explicitly use FORMAT, then the column's width will always be at least the value of SET NUMWIDTH.

If a value does not fit within the column width, SQL*Plus indicates overflow by displaying a pound sign (#) in place of each digit the width allows.

If a positive value is extremely large and a numeric overflow occurs when rounding a number, then the infinity sign (~) replaces the value. Likewise, if a negative value is extremely small and a numeric overflow occurs when rounding a number, then the negative infinity sign replaces the value (–~).

	With all number formats, SQL*Plus rounds each value to the specified number of significant digits as set with the SET NUMWIDTH command.
`HEA[DING]` *text*	Defines a column heading. If you do not use a HEADING clause, the column's heading defaults to *column* or *expr*. If *text* contains blanks or punctuation characters, you must enclose it with single or double quotes. Each occurrence of the HEADSEP character (by default, '│') begins a new line. For example,

```
COLUMN ENAME HEADING 'Employee |Name'
```

would produce a two-line column heading. See the HEADSEP variable of the SET command in this chapter for information on changing the HEADSEP character.

`JUS[TIFY] {L[EFT]|C[ENTER]|C[ENTRE]|R[IGHT]}`	
	Aligns the heading. If you do not use a JUSTIFY clause, headings for NUMBER columns default to RIGHT and headings for other column types default to LEFT.

`LIKE {`*expr*`|`*alias*`}`	
	Copies the display attributes of another column or expression (whose attributes you have already defined with another COLUMN command). LIKE copies only attributes not defined by another clause in the current COLUMN command.

`NEWL[INE]`	Starts a new line before displaying the column's value. NEWLINE has the same effect as FOLD_BEFORE.

`NEW_V[ALUE]` *variable*	
	Specifies a variable to hold a column value. You can reference the variable in TTITLE commands. Use NEW_VALUE to display column values or the date in the top title. You must include the column in a BREAK command with the SKIP PAGE action. The variable name cannot contain a pound sign (#).
	NEW_VALUE is useful for master/detail reports in which there is a new master record for each page. For master/detail reporting, you must also include the column in the ORDER BY clause. See the example at the end of this command description.

For information on displaying a column value in the bottom title, see COLUMN OLD_VALUE. Refer to TTITLE for more information on referencing variables in titles. See COLUMN FORMAT for details on formatting and valid format models.

NOPRI[NT]|PRI[NT]
Controls the printing of the column (the column heading and all the selected values). NOPRINT turns the printing of the column off. PRINT turns the printing of the column on.

NUL[L] *text*
Controls the text SQL*Plus displays for null values in the given column. The default is a white space. SET NULL controls the text displayed for all null values for all columns, unless overridden for a specific column by the NULL clause of the COLUMN command. When a NULL value is SELECTed, a variable's type will always become CHAR so the SET NULL text can be stored in it.

OLD_V[ALUE] *variable*
Specifies a variable to hold a column value. You can reference the variable in BTITLE commands. Use OLD_VALUE to display column values in the bottom title. You must include the column in a BREAK command with the SKIP PAGE action.

OLD_VALUE is useful for master/detail reports in which there is a new master record for each page. For master/detail reporting, you must also include the column in the ORDER BY clause.

For information on displaying a column value in the top title, see COLUMN NEW_VALUE. Refer to TTITLE for more information on referencing variables in titles.

ON|OFF
Controls the status of display attributes for a column. OFF disables the attributes for a column without affecting the attributes' definition. ON reinstates the attributes.

WRA[PPED]|WOR[D_WRAPPED]|TRU[NCATED]

Specifies how SQL*Plus will treat a datatype or DATE string that is too wide for a column. WRAPPED wraps the string within the column bounds, beginning new lines when required. When WORD_WRAP is enabled, SQL*Plus left justifies each new line, skipping all leading whitespace (for example, returns, newline characters, tabs and spaces), including embedded newline characters. Embedded whitespace not on a line boundary is not skipped. TRUNCATED truncates the string at the end of the first line of display.

Usage Notes   You can enter any number of COLUMN commands for one or more columns. All column attributes set for each column remain in effect for the remainder of the session, until you turn the column OFF, or until you use the CLEAR COLUMN command. Thus, the COLUMN commands you enter can control a column's display attributes for multiple SQL SELECT commands.

When you enter multiple COLUMN commands for the same column, SQL*Plus applies their clauses collectively. If several COLUMN commands apply the same clause to the same column, the last one entered will control the output.

Examples   To make the ENAME column 20 characters wide and display EMPLOYEE NAME on two lines at the top, enter

```
SQL> COLUMN ENAME FORMAT A20 HEADING 'EMPLOYEE |NAME'
```

To format the SAL column so that it shows millions of dollars, rounds to cents, uses commas to separate thousands, and displays $0.00 when a value is zero, you would enter

```
SQL> COLUMN SAL FORMAT $9,999,990.99
```

To assign the alias NET to a column containing a long expression, to display the result in a dollar format, and to display <NULL> for null values, you might enter

```
SQL> COLUMN SAL+COMM+BONUS-EXPENSES-INS-TAX ALIAS NET
SQL> COLUMN NET FORMAT $9,999,999.99 NULL '<NULL>'
```

Note that the example divides this column specification into two commands. The first defines the alias NET, and the second uses NET to define the format.

Also note that in the first command you must enter the expression exactly as you entered it (or will enter it) in the SELECT command.

Otherwise, SQL*Plus cannot match the COLUMN command to the appropriate column.

To wrap long values in a column named REMARKS, you can enter

```
SQL> COLUMN REMARKS FORMAT A20 WRAP
```

For example:

```
CUSTOMER   DATE      QUANTITY REMARKS
---------- --------- -------- --------------------
123        25-AUG-86      144 This order must be s
                               hipped by air freigh
                               t to ORD
```

If you replace WRAP with WORD_WRAP, REMARKS looks like this:

```
CUSTOMER   DATE      QUANTITY REMARKS
---------- --------- -------- --------------------
123        25-AUG-86      144 This order must be
                               shipped by air freight
                               to ORD
```

If you specify TRUNCATE, REMARKS looks like this:

```
CUSTOMER   DATE      QUANTITY REMARKS
---------- --------- -------- --------------------
123        25-AUG-86      144 This order must be s
```

In order to print the current date and the name of each job in the top title, enter the following. (For details on creating a date variable, see "Displaying the Current Date in Titles" under "Defining Page Titles and Dimensions" in Chapter 4.)

```
SQL> COLUMN JOB NOPRINT NEW_VALUE JOBVAR
SQL> COLUMN TODAY  NOPRINT NEW_VALUE DATEVAR
SQL> BREAK ON JOB SKIP PAGE ON TODAY
SQL> TTITLE CENTER 'Job Report' RIGHT DATEVAR  SKIP 2 -
>    LEFT 'Job:    ' JOBVAR SKIP 2
SQL> SELECT TO_CHAR(SYSDATE, 'MM/DD/YY') TODAY,
  2  ENAME, JOB, MGR, HIREDATE, SAL, DEPTNO
  3  FROM EMP WHERE JOB IN ('CLERK', 'SALESMAN')
  4  ORDER BY JOB, ENAME;
```

Your two page report would look similar to the following report, with "Job Report" centered within your current linesize:

```
                      Job Report              10/01/96

Job:    CLERK

ENAME         MGR HIREDATE       SAL    DEPTNO
---------- ------- --------- ----------- ----------
ADAMS        7788 14-JAN-87      1100        20
JAMES        7698 03-DEC-81       950        30
MILLER       7782 23-JAN-82      1300        10
SMITH        7902 17-DEC-80       800        20


                      Job Report              10/01/96

Job:    CLERK

ENAME         MGR HIREDATE       SAL    DEPTNO
---------- ------- --------- ----------- ----------
ALLEN        7698 20-JAN-81      1600        30
MARTIN       7698 03-DEC-81       950        30
MILLER       7782 23-JAN-82      1300        10
SMITH        7902 17-DEC-80       800        20
```

To change the default format of DATE columns to 'YYYY–MM–DD', you can enter

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD';
```

The following output results:

```
Session altered
```

To display the change, enter a SELECT statement, such as:

```
SQL> SELECT HIREDATE
  2  FROM EMP
  3  WHERE EMPNO = 7839;
```

The following output results:

```
HIREDATE
----------
1981-11-17
```

See the *Oracle8 Server SQL Reference Manual* for information on the ALTER SESSION command.

Note that in a SELECT statement, some SQL calculations or functions, such as TO_CHAR, may cause a column to be very wide. In such cases, use the FORMAT option to alter the column width.

COMPUTE

Purpose Calculates and prints summary lines, using various standard computations, on subsets of selected rows, or lists all COMPUTE definitions. (For details on how to create summaries, see "Clarifying Your Report with Spacing and Summary Lines" in Chapter 4.)

Syntax
```
COMP[UTE] [function [LAB[EL] text] ...
   OF {expr|column|alias} ...
   ON {expr|column|alias|REPORT|ROW} ...]
```

Terms and Clauses Refer to the following list for a description of each term or clause:

function ... Represents one of the functions listed in Table 6–2. If you specify more than one function, use spaces to separate the functions.

Function	*Computes*	*Applies to Datatypes*
AVG	Average of non-null values	NUMBER
COU[NT]	Count of non-null values	all types
MAX[IMUM]	Maximum value	NUMBER, CHAR, NCHAR, VARCHAR2 (VARCHAR), NVARCHAR2 (NCHAR VARYING)
MIN[IMUM]	Minimum value	NUMBER, CHAR, NCHAR, VARCHAR2 (VARCHAR), NVARCHAR2 (NCHAR VARYING)
NUM[BER]	Count of rows	all types
STD	Standard deviation of non-null values	NUMBER
SUM	Sum of non-null values	NUMBER
VAR[IANCE]	Variance of non-null values	NUMBER

Table 7 – 2 COMPUTE Functions

LAB[EL] *text* Defines the label to be printed for the computed value. If no LABEL clause is used, *text* defaults to the unabbreviated function keyword. If *text* contains spaces or punctuation, you must enclose it with single quotes. The label prints left justified and truncates to the column width or linesize, whichever is smaller. The maximum length of a label is 500 characters.

The label for the computed value appears in the break column specified. To suppress the label, use the NOPRINT option of the COLUMN command on the break column.

If you repeat a function in a COMPUTE command, SQL*Plus issues a warning and uses the first occurrence of the function.

With ON REPORT and ON ROW computations, the label appears in the first column listed in the SELECT statement. The label can be suppressed by using a NOPRINT column first in the SELECT statement. When you compute a function of the first column in the SELECT statement ON REPORT or ON ROW, then the computed value appears in the first column and the label is not displayed. To see the label, select a dummy column first in the SELECT list.

OF {*expr*|*column*|*alias*}...

Specifies the column(s) or expression(s) you wish to use in the computation. (*column* cannot have a table or view appended to it. To achieve this, you can alias the column in the SQL statement.) You must also specify these columns in the SQL SELECT command, or SQL*Plus will ignore the COMPUTE command.

If you use a SQL SELECT list alias, you must use the SQL alias in the COMPUTE command, not the column name. If you use the column name in this case, SQL*Plus will ignore the COMPUTE command.

If you do not want the computed values of a column to appear in the output of a SELECT command, specify that column in a COLUMN command with a NOPRINT clause. Use spaces to separate multiple expressions, columns, or aliases within the OF clause.

In the OF clause, you can refer to an expression or function reference in the SELECT statement by placing the expression or function reference in double quotes. Column names and aliases do not need quotes.

```
ON {expr|column|alias|REPORT|ROW} ...
```
Specifies the event SQL*Plus will use as a break. (*column* cannot have a table or view appended to it. To achieve this, you can alias the column in the SQL statement.) COMPUTE prints the computed value and restarts the computation when the event occurs (that is, when the value of the expression changes, a new ROW is fetched, or the end of the report is reached).

If multiple COMPUTE commands reference the same column in the ON clause, only the last COMPUTE command applies.

To reference a SQL SELECT expression or function reference in an ON clause, place the expression or function reference in quotes. Column names and aliases do not need quotes.

Enter COMPUTE without clauses to list all COMPUTE definitions.

Usage Notes In order for the computations to occur, the following conditions must all be true:

- One or more of the expressions, columns, or column aliases you reference in the OF clause must also be in the SELECT command.

- The expression, column, or column alias you reference in the ON clause must occur in the SELECT command and in the most recent BREAK command.

- If you reference either ROW or REPORT in the ON clause, also reference ROW or REPORT in the most recent BREAK command.

To remove all COMPUTE definitions, use the CLEAR COMPUTES command.

Examples To subtotal the salary for the "clerk", "analyst", and "salesman" job classifications with a compute label of "TOTAL", enter

```
SQL> BREAK ON JOB SKIP 1
SQL> COMPUTE SUM LABEL 'TOTAL' OF SAL ON JOB
SQL> SELECT JOB, ENAME, SAL
  2  FROM EMP
  3  WHERE JOB IN ('CLERK', 'ANALYST', 'SALESMAN')
  4  ORDER BY JOB, SAL;
```

The following output results:

```
JOB         ENAME             SAL
---------   ----------  ----------
ANALYST     SCOTT             3000
            FORD              3000
*********               ----------
TOTAL                         6000

CLERK       SMITH              800
            JAMES              950
            ADAMS             1100
            MILLER            1300
*********               ----------
TOTAL                         4150

SALESMAN    WARD              1250
            MARTIN            1250
            TURNER            1500
            ALLEN             1600
*********               ----------
TOTAL                         5600
```

To calculate the total of salaries less than 1,000 on a report, enter

```
SQL> COMPUTE SUM OF SAL ON REPORT
SQL> BREAK ON REPORT
SQL> COLUMN DUMMY HEADING ''
SQL> SELECT '   ' DUMMY, SAL, EMPNO
  2  FROM EMP
  3  WHERE SAL < 1000
  4  ORDER BY SAL;
```

The following output results:

```
            SAL       EMPNO
--- ----------  ----------
            800        7369
            950        7900
    ----------
sum      5350
```

To compute the average and maximum salary for the accounting and
sales departments, enter

```
SQL> BREAK ON DNAME SKIP 1
SQL> COMPUTE AVG LABEL 'Dept Average' –
>           MAX LABEL 'Dept Maximum' –
>       OF SAL ON DNAME
SQL> SELECT DNAME, ENAME, SAL
  2  FROM DEPT, EMP
  3  WHERE DEPT.DEPTNO = EMP.DEPTNO
  4  AND DNAME IN ('ACCOUNTING', 'SALES')
  5  ORDER BY DNAME;
```

The following output results:

```
DNAME           ENAME            SAL
------------- ---------- ----------
ACCOUNTING    CLARK           2450
              KING            5000
              MILLER          1300
**************            ----------
Dept Average             2916.66667
Dept Maximum                   5000

SALES         ALLEN           1600
              WARD            1250
              JAMES            950
              TURNER          1500
              MARTIN          1250
              BLAKE           2850
**************            ----------
Dept Average             1566.66667
Dept Maximum                   2850
```

To compute the sum of salaries for departments 10 and 20 without
printing the compute label:

```
SQL> COLUMN DUMMY NOPRINT
SQL> COMPUTE SUM OF SAL ON DUMMY
SQL> BREAK ON DUMMY SKIP 1
SQL> SELECT DEPTNO DUMMY, DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO <= 20
  4  ORDER BY DEPTNO;
```

SQL*Plus displays the following output:

```
    DEPTNO ENAME             SAL
---------- ---------- ----------
        10 KING             5000
        10 CLARK            2450
        10 MILLER           1300
                      ----------
                            8750


        20 JONES            2975
        20 FORD             3000
        20 SMITH             800
        20 SCOTT            3000
        20 ADAMS            1100
                      ----------
                           10875
```

If, instead, you do not want to print the label, only the salary total at
the end of the report:

```
SQL> COLUMN DUMMY NOPRINT
SQL> COMPUTE SUM OF SAL ON DUMMY
SQL> BREAK ON DUMMY
SQL> SELECT NULL DUMMY, DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO <= 20
  4  ORDER BY DEPTNO;
```

SQL*Plus displays the following output:

```
    DEPTNO ENAME             SAL
---------- ---------- ----------
        10 KING             5000
        10 CLARK            2450
        10 MILLER           1300
        20 JONES            2975
        20 FORD             3000
        20 SMITH             800
        20 SCOTT            3000
        20 ADAMS            1100
                      ----------
                           19625
```

CONNECT

Purpose   Connects a given username to Oracle.

Syntax   `CONN[ECT] [logon]`

where:

`logon`	Requires the following syntax: `username[/password][@database_specification]|/`

Terms and Clauses   Refer to the following list for a description of each term or clause:

`username [/password]`	Represent the username and password with which you wish to connect to Oracle. If you omit *username* and *password*, SQL*Plus prompts you for them. If you enter a slash (/) or simply enter [Return] to the prompt for *username*, SQL*Plus logs you in using a default logon (see "/" below).
	If you omit only *password,* SQL*Plus prompts you for *password.* When prompting, SQL*Plus does not display *password* on your terminal screen. See the PASSWORD command in this chapter for information about changing your password.
`database specification`	Consists of a SQL*Net connection string. The exact syntax depends upon the SQL*Net communications protocol your Oracle installation uses. For more information, refer to the SQL*Net manual appropriate for your protocol or contact your DBA. SQL*Plus does not prompt for a database specification, but uses your default database if you do not include a specification.
`/`	Represents a default logon using operating system authentication. You cannot enter a *database_specification* if you use a default logon. In a default logon, SQL*Plus typically attempts to log you in using the username OPS$*name*, where *name* is your operating system username. See the *Oracle8 Server Administrator's Guide* for information about operating system authentication.

Usage Notes   CONNECT commits the current transaction to the database, disconnects the current username from Oracle, and reconnects with the specified username.

If you log on or connect as a user whose account has expired, SQL*Plus prompts you to change your password before you can connect.

If an account is locked, a message is displayed and connection into that account (as that user) is not permitted until the account is unlocked by your DBA.

For further information on account management, refer to the documentation on the CREATE and ALTER USER commands. Also see the CREATE PROFILE command in the *Oracle8 Server SQL Reference Manual.*

Examples   To connect across SQL*Net using username SCOTT and password TIGER to the database known by the SQL*Net alias as FLEETDB, enter

```
SQL> CONNECT SCOTT/TIGER@FLEETDB
```

To connect using username SCOTT, letting SQL*Plus prompt you for the password, enter

```
SQL> CONNECT SCOTT
```

COPY

Purpose   Copies the data from a query to a table in a local or remote database.

Syntax
```
COPY {FROM username[/password]@database_specification|
    TO username[/password]@database_specification|
    FROM username[/password]@database_specification
    TO username[/password]@database_specification}
    {APPEND|CREATE|INSERT|REPLACE} destination_table
    [(column, column, column ...)] USING query
```

Terms and Clauses   Refer to the following list for a description of each term or clause:

username[/password]

> Represent the Oracle *username/password* you wish to COPY FROM and TO. In the FROM clause, *username/password* identifies the source of the data; in the TO clause, *username/password* identifies the destination. If you do not specify *password* in either the FROM clause or the TO clause, SQL*Plus will prompt you for it. SQL*Plus suppresses the display of your response to these prompts.

database_specification

> Consists of a SQL*Net connection string. You must include a *database_specification* clause in the COPY command. In the FROM clause*, database_specification* represents the database at the source; in the TO clause, *database_specification* represents the database at the destination. The exact syntax depends upon the SQL*Net communications protocol your Oracle installation uses. For more information, refer to the SQL*Net manual appropriate for your protocol or contact your DBA.

destination_table

> Represents the table you wish to create or to which you wish to add data.

(column, column, column, ...)

> Specifies the names of the columns in *destination_table*. You must enclose a name in double quotes if it contains lowercase letters or blanks.

If you specify columns, the number of columns must equal the number of columns selected by the query. If you do not specify any columns, the copied columns will have the same names in the destination table as they had in the source if COPY creates *destination_table*.

USING *query*   Specifies a SQL query (SELECT command) determining which rows and columns COPY copies.

FROM *username*[*/password*]@*database_specification*
        Specifies the username, password, and database that contains the data to be copied. If you omit the FROM clause, the source defaults to the database to which SQL*Plus is connected (that is, the database that other commands address). You must include a FROM clause to specify a source database other than the default.

TO *username*[*/password*]@*database_specification*
        Specifies the database containing the destination table. If you omit the TO clause, the destination defaults to the database to which SQL*Plus is connected (that is, the database that other commands address). You must include a TO clause to specify a destination database other than the default.

APPEND   Inserts the rows from *query* into *destination_table* if the table exists. If *destination_table* does not exist, COPY creates it.

CREATE   Inserts the rows from *query* into *destination_table* after first creating the table. If *destination_table* already exists, COPY returns an error.

INSERT   Inserts the rows from *query* into *destination_table*. If *destination_table* does not exist, COPY returns an error. When using INSERT, the USING *query* must select one column for each column in the *destination_table*.

REPLACE   Replaces *destination_table* and its contents with the rows from *query*. If *destination_table* does not exist, COPY creates it. Otherwise, COPY drops the existing table and replaces it with a table containing the copied data.

Usage Notes   To enable the copying of data between Oracle and non-Oracle databases, NUMBER columns are changed to DECIMAL columns in the destination table. Hence, if you are copying between Oracle databases, a NUMBER column with no precision will be changed to a DECIMAL(38) column. When copying between Oracle databases, you should use SQL commands (CREATE TABLE AS and INSERT) or you should ensure that your columns have a precision specified.

The SQL*Plus SET variable LONG limits the length of LONG columns that you copy. If any LONG columns contain data longer than the value of LONG, COPY truncates the data.

SQL*Plus performs a commit at the end of each successful COPY. If you set the SQL*Plus SET variable COPYCOMMIT to a positive value *n,* SQL*Plus performs a commit after copying every *n* batches of records. The SQL*Plus SET variable ARRAYSIZE determines the size of a batch.

Some operating environments require that database specifications be placed in double quotes.

Examples   The following command copies the entire EMP table to a table named WESTEMP. Note that the tables are located in two different databases. If WESTEMP already exists, SQL*Plus replaces the table and its contents. The columns in WESTEMP have the same names as the columns in the source table, EMP.

```
SQL> COPY FROM SCOTT/TIGER@HQ TO JOHN/CHROME@WEST -
> REPLACE WESTEMP -
> USING SELECT * FROM EMP
```

The following command copies selected records from EMP to the database to which SQL*Plus is connected. SQL*Plus creates SALESMEN through the copy. SQL*Plus copies only the columns EMPNO and ENAME, and at the destination names them EMPNO and SALESMAN.

```
SQL> COPY FROM SCOTT/TIGER@HQ -
> CREATE SALESMEN (EMPNO,SALESMAN) -
> USING SELECT EMPNO, ENAME FROM EMP -
> WHERE JOB='SALESMAN'
```

DEFINE

Purpose
Specifies a user variable and assigns it a CHAR value, or lists the value and variable type of a single variable or all variables.

Syntax
`DEF[INE] [variable]|[variable = text]`

Terms and Clauses
Refer to the following list for a description of each term or clause:

`variable`
Represents the user variable whose value you wish to assign or list.

`text`
Represents the CHAR value you wish to assign to *variable*. Enclose *text* in single quotes if it contains punctuation or blanks.

`variable = text`
Defines (names) a user variable and assigns it a CHAR value.

Enter DEFINE followed by *variable* to list the value and type of *variable*. Enter DEFINE with no clauses to list the values and types of all user variables.

Usage Notes
DEFINEd variables retain their values until one of the following events occurs:

- you enter a new DEFINE command referencing the variable

- you enter an UNDEFINE command referencing the variable

- you enter an ACCEPT command referencing the variable

- you reference the variable in the NEW_VALUE or OLD_VALUE clause of the COLUMN command and reference the column in a subsequent SQL SELECT command

- you EXIT SQL*Plus

Whenever you run a stored query or command file, SQL*Plus substitutes the value of *variable* for each substitution variable referencing *variable* (in the form &*variable* or &&*variable*). SQL*Plus will not prompt you for the value of *variable* in this session until you UNDEFINE *variable*.

Note that you can use DEFINE to define the variable, _EDITOR, which establishes the host system editor invoked by the SQL*Plus EDIT command.

If you continue the value of a DEFINEd variable on multiple lines (using the SQL*Plus command continuation character), SQL*Plus replaces each continuation character and carriage return you enter with a space in the resulting variable. For example, SQL*Plus interprets

```
SQL> DEFINE TEXT = 'ONE-
> TWO-
> THREE'
```

as

```
SQL> DEFINE TEXT = 'ONE TWO THREE'
```

Examples   To assign the value MANAGER to the variable POS, type:

```
SQL> DEFINE POS = MANAGER
```

If you execute a command that contains a reference to &POS, SQL*Plus will substitute the value MANAGER for &POS and will not prompt you for a POS value.

To assign the CHAR value 20 to the variable DEPTNO, type:

```
SQL> DEFINE DEPTNO = 20
```

Even though you enter the number 20, SQL*Plus assigns a CHAR value to DEPTNO consisting of two characters, 2 and 0.

To list the definition of DEPTNO, enter

```
SQL> DEFINE DEPTNO
DEFINE DEPTNO          = "20" (CHAR)
```

This result shows that the value of DEPTNO is 20.

DEL

Purpose   Deletes one or more lines of the buffer.

Syntax   `DEL [n|n m|n *|n LAST|*|* n|* LAST|LAST]`

Terms and Clauses   Refer to the following list for a description of each term or clause:

`n`	Deletes line *n*.
`n m`	Deletes lines *n* through *m*.
`n *`	Deletes line *n* through the current line.
`n LAST`	Deletes line *n* through the last line.
`*`	Deletes the current line.
`* n`	Deletes the current line through line *n*.
`* LAST`	Deletes the current line through the last line.
`LAST`	Deletes the last line.

Enter DEL with no clauses to delete the current line of the buffer.

Usage Notes   DEL makes the following line of the buffer (if any) the current line. You can enter DEL several times to delete several consecutive lines.

> **Note:** DEL is a SQL*Plus command and DELETE is a SQL command. For more information about the SQL DELETE command, see the *Oracle8 Server SQL Reference Manual.*

Examples   Assume the SQL buffer contains the following query:

```
1   SELECT ENAME, DEPTNO
2   FROM EMP
3   WHERE JOB = 'SALESMAN'
4*  ORDER BY DEPTNO
```

To make the line containing the WHERE clause the current line, you could enter

```
SQL> LIST 3
  3* WHERE JOB = 'SALESMAN'
```

followed by

```
SQL> DEL
```

The SQL buffer now contains the following lines:

```
1  SELECT ENAME, DEPTNO
2  FROM EMP
3* ORDER BY DEPTNO
```

To delete the second line of the buffer, enter

```
SQL> DEL 2
```

The SQL buffer now contains the following lines:

```
1  SELECT ENAME, DEPTNO
2* ORDER BY DEPTNO
```

DESCRIBE

Purpose Lists the column definitions for the specified table, view, or synonym or the specifications for the specified function or procedure.

Syntax `DESC[RIBE] {[`*`schema.`*`]`*`object`*`[`*`@database_link_name`*`]}`

Terms and Clauses Refer to the following list for a description of each term or clause:

schema	Represents the schema where the *object* resides. If you omit *schema*, SQL*Plus assumes you own *object*.
object	Represents the table, view, type, procedure, function, package or synonym you wish to describe.
database_link_name	Consists of the database link name corresponding to the database where *object* exists. For more information on which privileges allow access to another table in a different schema, refer to the *Oracle8 Server SQL Reference Manual.*

Usage Notes The description for tables, views, types and synonyms contains the following information:

- each column's name

- whether or not null values are allowed (NULL or NOT NULL) for each column

- datatype of columns, for example, NUMBER, CHAR, VARCHAR2 (VARCHAR), LONG, DATE, MLSLABEL, RAW, LONGRAW, or ROWID

- precision of columns (and scale, if any, for a numeric column)

When you do a DESCRIBE, VARCHAR columns are returned with a type of VARCHAR2.

The description for functions and procedures contains the following information:

- the type of PL/SQL object (function or procedure)

- the name of the function or procedure

- the type of value returned (for functions)

- the argument names, types, whether they are input or output, and default values, if any

Example   To describe the table EMP, enter

```
SQL> DESCRIBE EMP
```

DESCRIBE lists the following information:

```
Name                            Null?    Type
------------------------------ -------- ------------
EMPNO                          NOT NULL NUMBER(4)
ENAME                                   CHAR(10)
JOB                                     JOB(9)
MGR                                     NUMBER(4)
HIREDATE                                DATE
SAL                                     NUMBER(7,2)
COMM                                    NUMBER(7,2)
DEPTNO                                  NUMBER(2)
```

To describe a procedure called CUSTOMER_LOOKUP, enter

```
SQL> DESCRIBE customer_lookup
```

DESCRIBE lists the following information:

```
PROCEDURE customer_lookup
Argument Name         Type     In/Out   Default?
--------------------- -------- -------- ---------
CUST_ID               NUMBER   IN
CUST_NAME             VARCHAR2 OUT
```

To create and describe the package APACK that contains the procedures aproc and bproc, enter

```
SQL> CREATE PACKAGE apack AS
  2  PROCEDURE aproc(P1 CHAR, P2 NUMBER);
  3  PROCEDURE bproc(P1 CHAR, P2 NUMBER);
  4  END apack;
  5  /
SQL> DESCRIBE apack
```

DESCRIBE lists the following information:

```
PROCEDURE aproc
Argument Name         Type      In/Out   Default?
--------------------- -------- -------- ---------
P1                    CHAR      IN
P2                    NUMBER    IN
PROCEDURE bproc
Argument Name         Type      In/Out   Default?
--------------------- -------- -------- ---------
P1                    CHAR      IN
P2                    NUMBER    IN
```

To create and describe the object type ADDRESS that contains the attributes STREET and CITY, enter

```
SQL> CREATE TYPE ADDRESS AS OBJECT
  2 ( STREET  VARCHAR2(20),
  3    CITY    VARCHAR2(20)
  4 );
  5 /
SQL> DESCRIBE address
```

DESCRIBE lists the following information:

```
Name                              Null?    Type
------------------------------- -------- ------------
STREET                                     VARCHAR2(20)
CITY                                       VARCHAR2(20)
```

To create and describe the object type EMPLOYEE that contains the attributes ENAME, EMPADDR, JOB and SAL, enter

```
SQL> CREATE TYPE EMPLOYEE AS OBJECT
  2 ( ENAME   VARCHAR2(30),
  3    EMPADDR  ADDRESS,
  4    JOB     VARCHAR2(20),
  5    SAL     NUMBER(7,2)
  6 );
  7 /
SQL> DESCRIBE employee
```

DESCRIBE lists the following information:

```
Name                                 Null?    Type
------------------------------ -------- ------------
ENAME                                         VARCHAR2(30)
EMPADDR                                        ADDRESS
JOB                                            VARCHAR2(20)
SAL                                            NUMBER(7,2)
```

To create and describe the object type addr_type as a table of the object type ADDRESS, enter

```
SQL> CREATE TYPE addr_type IS TABLE OF ADDRESS;
  2  /
SQL> DESCRIBE addr_type
```

DESCRIBE lists the following information:

```
addr_type TABLE OF ADDRESS
Name                                 Null?    Type
------------------------------ -------- ------------
STREET                                         VARCHAR2(20)
CITY                                           VARCHAR2(20)
```

To create and describe the object type addr_varray as a varray of the object type ADDRESS, enter

```
SQL> CREATE TYPE addr_varray AS VARRAY(10) OF ADDRESS;
  2  /
SQL> DESCRIBE addr_varray
```

DESCRIBE lists the following information:

```
addr_varray VARRAY(10) OF ADDRESS
Name                                 Null?    Type
------------------------------ -------- ------------
STREET                                         VARCHAR2(20)
CITY                                           VARCHAR2(20)
```

To create and describe the table dept_emp that contains the columns DEPTNO, PERSON and LOC, enter

```
SQL> CREATE TABLE dept_emp
  2  ( DEPTNO  NUMBER,
  3    PERSON  EMPLOYEE,
  4    LOC     NUMBER
  5  );
  6  /
SQL> DESCRIBE dept_emp
```

DESCRIBE lists the following information:

```
Name                               Null?   Type
------------------------------ -------- ------------
DEPTNO                                   NUMBER
PERSON                                   EMPLOYEE
LOC                                      NUMBER
```

To create and describe the object type rational that contains the attributes NUMERATOR and DENOMINATOR, and the METHOD rational_order, enter

```
SQL> CREATE OR REPLACE TYPE rational AS OBJECT
  2 ( NUMERATOR   NUMBER,
  3    DENOMINATOR NUMBER,
  4    MAP MEMBER FUNCTION rational_order –
>      RETURN DOUBLE PRECISION,
  5    PRAGMA RESTRICT_REFERENCES
  6   (rational_order, RNDS, WNDS, RNPS, WNPS) );
  7  /
SQL> CREATE OR REPLACE TYPE BODY rational AS OBJECT
  2  MAP MEMBER FUNCTION rational_order –
>    RETURN DOUBLE PRECISION IS
  3   BEGIN
  4    RETURN NUMERATOR/DENOMINATOR;
  5   END;
  6  END;
  7  /
SQL> DESCRIBE rational
```

DESCRIBE lists the following information:

```
Name                               Null?   Type
------------------------------ -------- ------------
NUMERATOR                                NUMBER
DENOMINATOR                              NUMBER


METHOD
------
MAP MEMBER FUNCTION RATIONAL_ORDER RETURNS NUMBER
```

For more information on using the CREATE TYPE command, see your *Oracle8 Server SQL Reference Manual.*

DISCONNECT

Purpose   Commits pending changes to the database and logs the current username out of Oracle, but does not exit SQL*Plus.

Syntax   `DISC[ONNECT]`

Usage Notes   Use DISCONNECT within a command file to prevent user access to the database when you want to log the user out of Oracle but have the user remain in SQL*Plus. Use EXIT or QUIT to log out of Oracle and return control to your host computer's operating system.

Example   Your command file might begin with a CONNECT command and end with a DISCONNECT, as shown below.

```
SQL> GET MYFILE
  1  CONNECT ...
     .
     .
     .
     .
15* DISCONNECT
```

EDIT

Purpose Invokes a host operating system text editor on the contents of the specified file or on the contents of the buffer.

Syntax `ED[IT] [file_name[.ext]]`

Terms and Clauses Refer to the following for a description of the term or clause:

`file_name[.ext]` Represents the file you wish to edit (typically a command file).

Enter EDIT with no filename to edit the contents of the SQL buffer with the host operating system text editor.

Usage Notes If you omit the file extension, SQL*Plus assumes the default command-file extension (normally SQL). For information on changing the default extension, see the SUFFIX variable of the SET command in this chapter.

If you specify a filename, SQL*Plus searches for the file in the current working directory. If SQL*Plus cannot find the file in the current working directory, it creates a file with the specified name.

The user variable, _EDITOR, contains the name of the text editor invoked by EDIT. You can change the text editor by changing the value of _EDITOR. See DEFINE for information about changing the value of a user variable. If _EDITOR is undefined, EDIT attempts to invoke the default host operating system editor.

EDIT alone places the contents of the SQL buffer in a file by default named AFIEDT.BUF (in your current working directory) and invokes the text editor on the contents of the file. If the file AFIEDT.BUF already exists, it is overwritten with the contents of the buffer. You can change the default filename by using the SET EDITFILE command. For more information about setting a default filename for the EDIT command, see the EDITFILE variable of the SET command in this chapter.

> **Note:** The default file, AFIEDT.BUF, may have a different name on some operating systems.

If you do not specify a filename and the buffer is empty, EDIT returns an error message.

To leave the editing session and return to SQL*Plus, terminate the editing session in the way customary for the text editor. When you leave the editor, SQL*Plus loads the contents of the file into the buffer.

Example   To edit the file REPORT with the extension SQL using your host
operating system text editor, enter

```
SQL> EDIT REPORT
```

EXECUTE

Purpose   Executes a single PL/SQL statement. The EXECUTE command is often useful when you want to execute a PL/SQL statement that references a stored procedure. For more information on PL/SQL, see your *PL/SQL User's Guide and Reference.*

Syntax   `EXEC[UTE]` *statement*

Terms and Clauses   Refer to the following for a description of the term or clause:

statement   Represents a PL/SQL statement.

Usage Notes   If your EXECUTE command cannot fit on one line because of the PL/SQL statement, use the SQL*Plus continuation character (a hyphen) as shown in the example below.

The length of the command and the PL/SQL statement cannot exceed the length defined by SET LINESIZE.

Examples   The following EXECUTE command assigns a value to a bind variable:

```
SQL> EXECUTE :n := 1
```

The following EXECUTE command runs a PL/SQL statement that references a stored procedure:

```
SQL> EXECUTE –
:ID := EMP_MANAGEMENT.HIRE('BLAKE','MANAGER','KING',2990,'SALES')
```

Note that the value returned by the stored procedure is being placed in a bind variable, `:ID`. For information on how to create a bind variable, see the VARIABLE command in this chapter.

EXIT

Purpose   Terminates SQL*Plus and returns control to the operating system.

Syntax   `{EXIT|QUIT} [`<u>`SUCCESS`</u>`|FAILURE|WARNING|`*`n`*`|`*`variable`*`|`
`:`*`BindVariable`*`] [`<u>`COMMIT`</u>`|ROLLBACK]`

Terms and Clauses   Refer to the following list for a description of each term or clause:

`{EXIT|QUIT}`	Can be used interchangeably (QUIT is a synonym for EXIT).
<u>`SUCCESS`</u>	Exits normally.
`FAILURE`	Exits with a return code indicating failure.
`WARNING`	Exits with a return code indicating warning.
<u>`COMMIT`</u>	Saves pending changes to the database before exiting.
`n`	Represents an integer you specify as the return code.
`variable`	Represents a user-defined or system variable (but not a bind variable), such as SQL.SQLCODE. EXIT *variable* exits with the value of *variable* as the return code.
`:BindVariable`	Represents a variable created in SQL*Plus with the VARIABLE command, and then referenced in PL/SQL, or other subprograms. *:BindVariable* exits the subprogram and returns you to SQL*Plus.
`ROLLBACK`	Executes a ROLLBACK statement and abandons pending changes to the database before exiting.

EXIT with no clauses commits and exits with a value of SUCCESS.

Usage Notes   EXIT allows you to specify an operating system return code. This allows you to run SQL*Plus command files in batch mode and to detect programmatically the occurrence of an unexpected event. The manner of detection is operating-system-specific. See the Oracle installation and user's manual(s) provided for your operating system for details.

The key words SUCCESS, WARNING, and FAILURE represent operating-system-dependent values. On some systems, WARNING and FAILURE may be indistinguishable.

> **Note:** SUCCESS, FAILURE, and WARNING are not reserved words.

The range of operating system return codes is also restricted on some operating systems. This limits the portability of EXIT *n* and EXIT *variable* between platforms. For example, on UNIX there is only one byte of storage for return codes; therefore, the range for return codes is limited to zero to 255.

If you make a syntax error in the EXIT options or use a non-numeric variable, SQL*Plus performs an EXIT FAILURE COMMIT.

For information on exiting conditionally, see the WHENEVER SQLERROR and WHENEVER OSERROR commands later in this chapter.

Example   The following example commits all uncommitted transactions and returns the error code of the last executed SQL command or PL/SQL block:

```
SQL> EXIT SQL.SQLCODE
```

The location of the return code depends on your system. Consult your DBA for information concerning how your operating system retrieves data from a program. See TTITLE in this chapter for more information on SQL.SQLCODE.

GET

Purpose	Loads a host operating system file into the SQL buffer.
Syntax	`GET file_name[.ext] [LIS[T]|NOL[IST]]`
Terms and Clauses	Refer to the following list for a description of each term or clause:

`file_name[.ext]`	Represents the file you wish to load (typically a command file).
`LIS[T]`	Lists the contents of the file.
`NOL[IST]`	Suppresses the listing.

Usage Note   If you do not specify a file extension, SQL*Plus assumes the default command-file extension (normally SQL). For information on changing the default extension, see the SUFFIX variable of the SET command in this chapter.

If part of the filename you are specifying contains the word *list* or the word *file*, you need to put the name in double quotes.

SQL*Plus searches for the file in the current working directory.

The operating system file should contain a single SQL statement or PL/SQL block. The statement should not be terminated with a semicolon.

If a SQL*Plus command or more than one SQL statement or PL/SQL block is loaded into the SQL buffer from an operating system file, an error occurs when the RUN or slash (/) command is used to execute the buffer.

The GET command can be used to load files created with the SAVE command. See the SAVE command in this chapter for more information.

Example   To load a file called YEARENDRPT with the extension SQL into the buffer, type

```
SQL> GET YEARENDRPT
```

HELP

Purpose: Accesses the SQL*Plus help system.

Syntax HELP [*topic*]

Terms and Clauses Refer to the following for a description of the term or clause:

topic   Represents a SQL*Plus help topic. This can be a SQL*Plus command (for example, COLUMN), a SQL statement (for example, INSERT), or a PL/SQL statement (for example, IF).

Enter HELP without *topic* to get help on the help system.

Usage Notes You can only enter one topic after HELP. You can abbreviate the topic (e.g., COL for COLUMN). However, if you enter only an abbreviated topic and the abbreviation is ambiguous, SQL*Plus will display help for all topics that match the abbreviation. For example, if you entered

```
SQL> HELP EX
```

SQL*Plus would display the syntax for the EXECUTE command followed by the syntax for the EXIT command.

If you get a response indicating that help is not available, consult your database administrator.

Example To see a list of SQL*Plus commands and PL/SQL and SQL statements, enter

```
SQL> HELP TOPICS
```

HOST

Purpose Executes a host operating system command without leaving SQL*Plus.

Syntax HO[ST] [*command*]

Terms and Clauses Refer to the following for a description of the term or clause:

command   Represents a host operating system command.

Enter HOST without *command* to display an operating system prompt. You can then enter multiple operating system commands. For information on returning to SQL*Plus, refer to the Oracle installation and user's manual(s) provided for your operating system.

Usage Notes With some operating systems, you can use a "$" (VMS), "!" (UNIX), or another character instead of HOST. See the Oracle installation and user's manual(s) provided for your operating system for details.

You may not have access to the HOST command, depending on your operating system. See the Oracle installation and user's manual(s) provided for your operating system or ask your DBA for more information.

SQL*Plus removes the SQLTERMINATOR (a semicolon by default) before the HOST command is issued. A workaround for this is to add another SQLTERMINATOR. See the SQLTERMINATOR variable of the SET command in this chapter for more information on the SQLTERMINATOR.

Example To execute an operating system command, ls *.sql, enter

```
SQL> HOST ls *.sql
```

INPUT

Purpose   Adds one or more new lines of text after the current line in the buffer.

Syntax   `I[NPUT] [text]`

Terms and Clauses   Refer to the following for a description of the term or clause:

> `text`   Represents the text you wish to add. To add a single line, enter the text of the line after the command INPUT, separating the text from the command with a space. To begin the line with one or more spaces, enter two or more spaces between INPUT and the first non-blank character of *text.*

To add several lines, enter INPUT with no *text.* INPUT prompts you for each line. To leave INPUT, enter a null (empty) line.

Usage Notes   If you enter a line number at the command prompt larger than the number of lines in the buffer, and follow the number with text, SQL*Plus adds the text in a new line at the end of the buffer. If you specify zero (0) for the line number and follow the zero with text, then SQL*Plus inserts the line at the beginning of the buffer (that line becomes line 1).

Examples   Assume the SQL buffer contains the following command:

```
1   SELECT ENAME, DEPTNO, SAL, COMM
2   FROM EMP
```

To add an ORDER BY clause to the query, enter

```
SQL> LIST 2
  2* FROM   EMP
SQL> INPUT ORDER BY ENAME
```

LIST 2 ensures that line 2 is the current line. INPUT adds a new line containing the ORDER BY clause after the current line. The SQL buffer now contains the following lines:

```
1   SELECT ENAME, DEPTNO, SAL, COMM
2   FROM   EMP
3* ORDER BY ENAME
```

To add a two-line WHERE clause, enter

```
SQL> LIST 2
  2* FROM EMP
SQL> INPUT
  3   WHERE JOB = 'SALESMAN'
  4   AND COMM  500
  5
```

INPUT prompts you for new lines until you enter an empty line. The SQL buffer now contains the following lines:

```
1   SELECT ENAME, DEPTNO, SAL, COMM
2   FROM EMP
3   WHERE JOB = 'SALESMAN'
4   AND COMM  500
5   ORDER BY ENAME
```

LIST

Purpose   Lists one or more lines of the SQL buffer.

Syntax   `L[IST] [n|n m|n *|n LAST|*|* n|* LAST|LAST]`

Terms and Clauses   Refer to the following list for a description of each term or clause:

`n`	Lists line *n*.
`n m`	Lists lines *n* through *m*.
`n *`	Lists line *n* through the current line.
`n LAST`	Lists line *n* through the last line.
`*`	Lists the current line.
`* n`	Lists the current line through line *n*.
`* LAST`	Lists the current line through the last line.
`LAST`	Lists the last line.

Enter LIST with no clauses to list all lines.

Usage Notes   The last line listed becomes the new current line (marked by an asterisk).

Example   To list the contents of the buffer, enter

```
SQL> LIST
```

You will see a listing of all lines in the buffer, similar in form to the following example:

```
1   SELECT ENAME, DEPTNO, JOB
2   FROM EMP
3   WHERE JOB = 'CLERK'
4* ORDER BY DEPTNO
```

The asterisk indicates that line 4 is the current line.

To list the second line only, enter

```
SQL> LIST 2
```

You will then see this:

```
2* FROM EMP
```

To list the current line (now line 2) to the last line, enter

```
SQL> LIST * LAST
```

You will then see this:

```
2   FROM EMP
3   WHERE JOB = 'CLERK'
4* ORDER BY DEPTNO
```

PASSWORD

Purpose   Allows you to change a password without echoing the password on an input device.

Syntax   `PASSW[ORD] [`*`username`*`]`

Terms and Clauses   Refer to the following for a description of the clause or term:

username   Specifies the user. If you do not specify a username, *username* defaults to the current user.

Usage Notes   To change the password of another user, you must have been granted the appropriate privilege.

For more information about changing your password, see the CONNECT command in this chapter.

Example   Suppose you are logged on as scott/tiger, and want to change the password to tigertiger

```
SQL> passw
Changing password for scott
Old password: tiger
New password: tigertiger
Retype new password: tigertiger
Password changed
SQL>
```

Suppose you are logged on as a DBA, and want to change the password for user usera (currently identified by passa) to passusera

```
SQL> passw usera
Changing password for usera
New password: passusera
Retype new password: passusera
Password changed
SQL>
```

PAUSE

Purpose   Displays an empty line followed by a line containing text, then waits for the user to press [Return], or displays two empty lines and waits for the user's response.

Syntax   `PAU[SE] [text]`

Terms and Clauses   Refer to the following for a description of the clause or term:

`text`   Represents the text you wish to display.

Enter PAUSE followed by no text to display two empty lines.

Usage Notes   Because PAUSE always waits for the user's response, it is best to use a message that tells the user explicitly to press [Return].

PAUSE reads input from the terminal (if a terminal is available) even when you have designated the source of the command input as a file.

For information on pausing between pages of a report, see the PAUSE variable of the SET command later in this chapter.

Example   To print "Adjust paper and press RETURN to continue." and to have SQL*Plus wait for the user to press [Return], you might include the following PAUSE command in a command file:

```
SET PAUSE OFF
PAUSE Adjust paper and press RETURN to continue.
SELECT ...
```

PRINT

Purpose Displays the current value of bind variables. For more information on bind variables, see your *PL/SQL User's Guide and Reference.*

Syntax `PRI[NT] [variable ...]`

Terms and Clauses Refer to the following for a description of the clause or term:

`variable ...` Represents the names of the bind variables whose values you wish to display.

Enter PRINT with no variables to print all bind variables.

Usage Notes Bind variables are created using the VARIABLE command. For more information and examples, see the VARIABLE command in this chapter.

You can control the formatting of the PRINT output just as you would query output. For more information, see the formatting techniques described in Chapter 4.

To automatically display bind variables referenced in a successful PL/SQL block or used in an EXECUTE command, use the AUTOPRINT clause of the SET command. For more information, see the SET command in this chapter.

Example The following example illustrates a PRINT command:

```
SQL> VARIABLE n NUMBER
SQL> BEGIN
  2   :n := 1;
  3   END;
SQL> PRINT n
         N
----------
         1
```

PROMPT

Purpose Sends the specified message or a blank line to the user's screen.

Syntax `PROMPT [text]`

Terms and Clauses Refer to the following for a description of the term or clause:

text   Represents the text of the message you wish to
               display. If you omit *text*, PROMPT displays a blank
               line on the user's screen.

Usage Notes You can use this command in command files to give information to the
               user.

Example The following example shows the use of PROMPT in conjunction with
            ACCEPT in a command file called ASKFORDEPT. ASKFORDEPT
            contains the following SQL*Plus and SQL commands:

```
PROMPT
PROMPT Please enter a valid department
PROMPT For example:  10, 20, 30, 40
ACCEPT NEWDEPT NUMBER PROMPT 'DEPT:> '
SELECT DNAME FROM DEPT
WHERE DEPTNO = &NEWDEPT
```

Assume you run the file using START or @:

```
SQL> @ASKFORDEPT
```

SQL*Plus displays the following prompts:

```
Please enter a valid department
For example:  10, 20, 30, 40
DEPT:>
```

You can enter a department number at the prompt `DEPT:>`. By default,
SQL*Plus lists the line containing `&NEWDEPT` before and after
substitution, and then displays the department name corresponding to
the number entered at the `DEPT:>` prompt.

REMARK

Purpose   Begins a comment in a command file. SQL*Plus does not interpret the comment as a command.

Syntax   `REM[ARK]`

Usage Notes   The REMARK command must appear at the beginning of a line, and the comment ends at the end of the line. A line cannot contain both a comment and a command.

For details on entering comments in command files using the SQL comment delimiters, /* ... */, or the ANSI/ISO comment delimiter, – – ..., refer to "Placing Comments in Command Files" in Chapter 3.

Example   The following command file contains some typical comments:

```
REM COMPUTE uses BREAK ON REPORT to break on end of table.
BREAK ON REPORT
COMPUTE SUM OF "DEPARTMENT 10" "DEPARTMENT 20" –
"DEPARTMENT 30" "TOTAL BY JOB" ON REPORT
REM Each column displays the sums of salaries by job for
REM one of the departments 10, 20, 30.
SELECT JOB,
       SUM( DECODE( DEPTNO, 10, SAL, 0)) "DEPARTMENT 10",
       SUM( DECODE( DEPTNO, 20, SAL, 0)) "DEPARTMENT 20",
       SUM( DECODE( DEPTNO, 30, SAL, 0)) "DEPARTMENT 30",
       SUM(SAL) "TOTAL BY JOB"
FROM EMP
GROUP BY JOB
```

REPFOOTER

Purpose	Places and formats a specified report footer at the bottom of each report, or lists the current REPFOOTER definition.
Syntax	REPF[OOTER] [PAGE] [*printspec* [*text*|*variable*] ...] | [<u>OFF</u>|ON]
Terms and Clauses	Refer to the REPHEADER command for additional information on terms and clauses in the REPFOOTER command syntax.
	Enter REPFOOTER with no clauses to list the current REPFOOTER definition.
Usage Notes	If you do not enter a *printspec* clause before the text or variables, REPFOOTER left justifies the text or variables.
	You can use any number of constants and variables in a *printspec*. SQL*Plus displays the constants and variables in the order you specify them, positioning and formatting each constant or variable as specified by the *printspec* clauses that precede it.

> **Note:** If SET EMBEDDED is ON, the report footer is suppressed.

Example To define "END EMPLOYEE LISTING REPORT" as a report footer on a separate page and to center it, enter:

```
SQL> REPFOOTER PAGE CENTER 'END EMPLOYEE LISTING REPORT'
SQL> TTITLE RIGHT 'Page: ' FORMAT 999 SQL.PNO
SQL> SELECT ENAME, SAL
  2   FROM EMP
  3   WHERE SAL > 2000;
                                                        Page: 1
ENAME            SAL
––––––––––  ––––––––––
JONES           2975
BLAKE           2850
CLARK           2450
SCOTT           3000
KING            5000
FORD            3000
                                                        Page: 2
                 END EMPLOYEE LISTING REPORT
```

To suppress the report footer without changing its definition, enter:

```
SQL> REPFOOTER OFF
```

REPHEADER

Purpose   Places and formats a specified report header at the top of each report, or lists the current REPHEADER definition.

Syntax   REPH[EADER] [PAGE] [*printspec* [*text*|*variable*] ...] |
      [<u>OFF</u>|ON]

where *printspec* represents one or more of the following clauses used to place and format the *text*:

```
COL n
S[KIP] [n]
TAB n
LE[FT]
CE[NTER]
R[IGHT]
BOLD
FORMAT text
```

Terms and Clauses   Refer to the following list for a description of each term or clause. These terms and clauses also apply to the REPFOOTER command.

PAGE   Begins a new page after printing the specified report header or before printing the specified report footer.

Note: You must specify SET NEWPAGE 0 to create a physical page break using this command.

text   Represents the report header or footer text. Enter *text* in single quotes if you want to place more than one word on a single line. The default is NULL.

variable   Represents a user variable or any of the following system-maintained values:

- SQL.LNO (current line number)

- SQL.PNO (current page number)

- SQL.RELEASE (current Oracle release number)

- SQL.SQLCODE (current error code)

- SQL.USER (current username)

To print one of these values, reference the appropriate variable in the report header or footer. You can format *variable* with the FORMAT clause.

OFF	Turns the report header or footer off (suppresses its display) without affecting its definition.
ON	Turns the report header or footer on (restores its display). When you define a report header or footer, SQL*Plus automatically sets REPHEADER or REPFOOTER to ON.
COL *n*	Indents to column *n* of the current line (backward if column *n* has been passed). "Column" in this context means print position, not table column.
S[KIP] [*n*]	Skips to the start of a new line *n* times; if you omit *n,* one time; if you enter zero for *n,* backward to the start of the current line.
TAB *n*	Skips forward *n* columns (backward if you enter a negative value for *n*). "Column" in this context means print position, not table column.
LE[FT], CE[NTER], and R[IGHT]	Left-align, center, and right-align data on the current line respectively. SQL*Plus aligns following data items as a group, up to the end of the *printspec* or the next LEFT, CENTER, RIGHT, or COL command. CENTER and RIGHT use the SET LINESIZE value to calculate the position of the data item that follows.
BOLD	Prints data in bold print. SQL*Plus represents bold print on your terminal by repeating the data on three consecutive lines. On some operating systems, SQL*Plus may instruct your printer to print bolded text on three consecutive lines, instead of bold.
FORMAT *text*	Specifies a format model that determines the format of following data items, up to the next FORMAT clause or the end of the command. The format model must be a *text* constant such as A10 or $999. See COLUMN FORMAT for more information on formatting and valid format models.
	If the datatype of the format model does not match the datatype of a given data item, the FORMAT clause has no effect on that item.

If no appropriate FORMAT model precedes a given data item, SQL*Plus prints NUMBER values according to the format specified by SET NUMFORMAT or, if you have not used SET NUMFORMAT, the default format. SQL*Plus prints DATE values according to the default format.

Refer to the FORMAT clause of the COLUMN command in this chapter for more information on default formats.

Enter REPHEADER with no clauses to list the current REPHEADER definition.

Usage Notes   If you do not enter a *printspec* clause before the text or variables, REPHEADER left justifies the text or variables.

You can use any number of constants and variables in a *printspec*. SQL*Plus displays the constants and variables in the order you specify them, positioning and formatting each constant or variable as specified by the *printspec* clauses that precede it.

Example   To define "EMPLOYEE LISTING REPORT" as a report header on a separate page, and to center it, enter:

```
SQL> REPHEADER PAGE CENTER 'EMPLOYEE LISTING REPORT'
SQL> TTITLE RIGHT 'Page: ' FORMAT 999 SQL.PNO
SQL> SELECT ENAME, SAL
  2  FROM EMP
  3  WHERE SAL > 2000;

                                                    Page: 1

                   EMPLOYEE LISTING REPORT

                                                    Page: 2

ENAME            SAL
---------- ----------
JONES           2975
BLAKE           2850
CLARK           2450
SCOTT           3000
KING            5000
FORD            3000
6 rows selected.
```

To suppress the report header without changing its definition, enter:

```
SQL> REPHEADER OFF
```

RUN

Purpose   Lists and executes the SQL command or PL/SQL block currently stored in the SQL buffer.

Syntax   `R[UN]`

Usage Notes   RUN causes the last line of the SQL buffer to become the current line.

The slash command (/) functions similarly to RUN, but does not list the command in the SQL buffer on your screen.

Example   Assume the SQL buffer contains the following query:

```
SELECT DEPTNO FROM DEPT
```

To RUN the query, enter

```
SQL> RUN
```

The following output results:

```
1* SELECT DEPTNO FROM DEPT

    DEPTNO
----------
        10
        20
        30
        40
```

SAVE

Purpose   Saves the contents of the SQL buffer in a host operating system file (a command file).

Syntax   `SAV[E]` *`file_name`*`[.ext]` `[`<u>`CRE[ATE]`</u>`|REP[LACE]|APP[END]]`

Terms and Clauses   Refer to the following list for a description of each term or clause:

`file_name``[.ext]`	Specifies the command file in which you wish to save the buffer's contents.
<u>`CRE[ATE]`</u>	Creates the file if the file does not exist.
`REP[LACE]`	Replaces the contents of an existing file. If the file does not exist, REPLACE creates the file.
`APP[END]`	Adds the contents of the buffer to the end of the file you specify.

Usage Notes   If you do not specify an extension, SQL*Plus assumes the default command-file extension (normally SQL). For information on changing this default extension, see the SUFFIX variable of the SET command in this chapter.

If you wish to SAVE a file under a name identical to a SAVE command clause (CREATE, REPLACE, or APPEND), you must specify a file extension.

When you SAVE the contents of the SQL buffer, SAVE adds a line containing a slash (/) to the end of the file.

If the filename you specify is the word *file*, you need to put the name in single quotes.

Example   To save the contents of the buffer in a file named DEPTSALRPT with the extension SQL, enter

```
SQL> SAVE DEPTSALRPT
```

To save the contents of the buffer in a file named DEPTSALRPT with the extension OLD, enter

```
SQL> SAVE DEPTSALRPT.OLD
```

SET

Purpose   Sets a system variable to alter the SQL*Plus environment for your current session, such as

- the display width for NUMBER data
- the display width for LONG data
- enabling or disabling the printing of column headings
- the number of lines per page

Syntax   SET *system_variable value*

where *system_variable value* represents a system variable followed by a value, as shown below:

```
APPI[NFO]{ON|OFF|text}
ARRAY[SIZE] {20|n}
AUTO[COMMIT] {OFF|ON|IMM[EDIATE]|n}
AUTOP[RINT] {OFF|ON}
AUTOT[RACE] {OFF|ON|TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
BLO[CKTERMINATOR] {.|c}
CMDS[EP] {;|c|OFF|ON}
COLSEP {_|text}
COM[PATIBILITY] {V7|V8|NATIVE}
CON[CAT] {.|c|OFF|ON}
COPYC[OMMIT] {0|n}
COPYTYPECHECK {OFF|ON}
DEF[INE] {'&'|c|OFF|ON}
ECHO {OFF|ON}
EDITF[ILE] file_name[.ext]
EMB[EDDED] {OFF|ON}
ESC[APE] {\|c|OFF|ON}
FEED[BACK] {6|n|OFF|ON}
FLAGGER {OFF|ENTRY|INTERMED[IATE]|FULL}
FLU[SH] {OFF|ON}
HEA[DING] {OFF|ON}
HEADS[EP] {||c|OFF|ON}
LIN[ESIZE] {80|n}
LOBOF[FSET] {n|1}
LONG {80|n}
LONGC[HUNKSIZE] {80|n}
NEWP[AGE] {1|n|NONE}
NULL text
NUMF[ORMAT] format
```

```
NUM[WIDTH] {10|n}
PAGES[IZE] {24|n}
PAU[SE] {OFF|ON|text}
RECSEP {WR[APPED]|EA[CH]|OFF}
RECSEPCHAR {_|c}
SERVEROUT[PUT] {OFF|ON} [SIZE n] [FOR[MAT] {WRA[PPED]|
    WOR[D_WRAPPED]|TRU[NCATED]}]
SHIFT[INOUT] {VIS[IBLE]|INV[ISIBLE]}
SHOW[MODE] {OFF|ON}
SQLC[ASE] {MIX[ED]|LO[WER]|UP[PER]}
SQLCO[NTINUE] {> |text}
SQLN[UMBER] {OFF|ON}
SQLPRE[FIX] {#|c}
SQLP[ROMPT] {SQL>|text}
SQLT[ERMINATOR] {;|c|OFF|ON}
SUF[FIX] {SQL|text}
TAB {OFF|ON}
TERM[OUT] {OFF|ON}
TI[ME] {OFF|ON}
TIMI[NG] {OFF|ON}
TRIM[OUT] {OFF|ON}
TRIMS[POOL] {ON|OFF}
UND[ERLINE] {-|c|ON|OFF}
VER[IFY] {OFF|ON}
WRA[P] {OFF|ON}
```

Terms and Clauses   Refer to the following list for a description of each term, clause, or
system variable:

```
APPI[NFO]{ON|OFF|text}
```
        Sets automatic registering of command files
through the DBMS_APPLICATION_INFO
package. This enables the performance and
resource usage of each command file to be
monitored by your DBA. The registered name
appears in the MODULE column of the
V$SESSION and V$SQLAREA virtual tables. You
can also read the registered name using the
DBMS_APPLICATION_INFO.READ_MODULE
procedure.

ON registers command files invoked by the @, @@
or START commands. OFF disables registering of
command files. Instead, the current value of *text* is

registered. *Text* specifies the text to register when no command file is being run or when APPINFO is OFF. The default for *text* is "SQL*Plus." If you enter multiple words for *text*, you must enclose them in quotes. The maximum length for *text* is limited by the DBMS_APPLICATION_INFO package.

The registered name has the format *nn@xfilename* where: *nn* is the depth level of command file; *x* is '<' when the command file name is truncated, otherwise, it is blank; and *filename* is the command file name, possibly truncated to the length allowed by the DBMS_APPLICATION_INFO package interface.

Note: To use this feature, you must have access to the DBMS_APPLICATION_INFO package. Run DBMSUTIL.SQL (this name may vary depending on your operating system) as SYS to create the DBMS_APPLICATION_INFO package. DBMSUTIL.SQL is part of the Oracle8 Server product.

For more information on the DBMS_APPLICATION_INFO package, see "Registering Applications" in the *Oracle8 Server Tuning* manual.

Note: APPINFO is not available with TRUSTED Oracle.

ARRAY[SIZE] {20|*n*}

Sets the number of rows—called a *batch*—that SQL*Plus will fetch from the database at one time. Valid values are 1 to 5000. A large value increases the efficiency of queries and subqueries that fetch many rows, but requires more memory. Values over approximately 100 provide little added performance. ARRAYSIZE has no effect on the results of SQL*Plus operations other than increasing efficiency.

AUTO[COMMIT] {OFF|ON|IMM[EDIATE]|*n*}

Controls when Oracle commits pending changes to the database. ON commits pending changes to the database after Oracle executes each successful INSERT, UPDATE, or DELETE command or PL/SQL block. OFF suppresses automatic

committing so that you must commit changes manually (for example, with the SQL command COMMIT). IMMEDIATE functions in the same manner as the ON option. *n* commits pending changes to the database after Oracle executes *n* successful SQL INSERT, UPDATE, or DELETE commands or PL/SQL blocks. *n* cannot be less than zero or greater than 2,000,000,000. The statement counter is reset to zero after successful completion of

- *n* INSERT, UPDATE or DELETE commands or PL/SQL blocks

- a commit

- a rollback

- a SET AUTOCOMMIT command

Note: For this feature, a PL/SQL block is considered one transaction, regardless of the actual number of SQL commands contained within it.

AUTOP[RINT] {<u>OFF</u>|ON}

Sets the automatic PRINTing of bind variables. ON or OFF controls whether SQL*Plus automatically displays bind variables (referenced in a successful PL/SQL block or used in an EXECUTE command). For more information about displaying bind variables, see the PRINT command in this chapter.

AUTOT[RACE] {<u>OFF</u>|ON|TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]

Displays a report on the execution of successful SQL DML statements (SELECT, INSERT, UPDATE or DELETE). The report can include execution statistics and the query execution path.

OFF does not display a trace report. ON displays a trace report. TRACEONLY displays a trace report, but does not print query data, if any. EXPLAIN shows the query execution path by performing an EXPLAIN PLAN. STATISTICS displays SQL statement statistics.

Using ON or TRACEONLY with no explicit options defaults to EXPLAIN STATISTICS.

The TRACEONLY option may be useful to suppress the query data of large queries. If

STATISTICS is specified, SQL*Plus still fetches the query data from the server, however, the data is not displayed.

The AUTOTRACE report is printed after the statement has successfully completed.

Information about Execution Plans and the statistics is documented in the *Oracle8 Server Tuning* manual.

To use the EXPLAIN option, you must first create the table PLAN_TABLE in your schema. The description of this table is specific to the version of the database to which you are connected. Use UTLXPLAN.SQL (this name may vary depending on your operating system) to create PLAN_TABLE. UTLXPLAN.SQL is part of the Oracle8 Server product. Contact your DBA if you cannot create this table.

To access STATISTICS data, you must have access to several Dynamic Performance tables (for information about the Dynamic Performance or "V$" tables, see the Oracle8 Server documentation). Access can be granted using the role created in PLUSTRCE.SQL (this name may vary depending on your operating system). You must run PLUSTRCE.SQL as SYS and grant the role to users who will use SET AUTOTRACE. Contact your DBA to perform these steps.

When SQL*Plus produces a STATISTICS report, a second connection to the database is automatically created. This connection is closed when the STATISTICS option is set to OFF, or you log out of SQL*Plus.

The formatting of your AUTOTRACE report may vary depending on the version of the server to which you are connected and the configuration of the server.

AUTOTRACE is not available when FIPS flagging is enabled, or with TRUSTED Oracle.

See "Tracing Statements" in Chapter 3 for more information on AUTOTRACE.

BLO[CKTERMINATOR] {.|c}

Sets the non-alphanumeric character used to end PL/SQL blocks to *c.* To execute the block, you must issue a RUN or / (slash) command.

CMDS[EP] {;|c|OFF|ON}

Sets the non-alphanumeric character used to separate multiple SQL*Plus commands entered on one line to *c.* ON or OFF controls whether you can enter multiple commands on a line; ON automatically sets the command separator character to a semicolon (;).

COLSEP {_|text}   Sets the text to be printed between SELECTed columns. If the COLSEP variable contains blanks or punctuation characters, you must enclose it with single quotes. The default value for *text* is a single space.

In multi-line rows, the column separator does not print between columns that begin on different lines. The column separator does not appear on blank lines produced by BREAK ... SKIP *n* and does not overwrite the record separator. See SET RECSEP in this chapter for more information.

COM[PATIBILITY] {V7|V8|NATIVE}

Specifies the version of Oracle to which you are currently connected. Set COMPATIBILITY to V7 for Oracle7, or V8 for Oracle8. Set COMPATIBILITY to NATIVE if you wish the database to determine the setting (for example, if connected to Oracle8, compatibility would default to V8). COMPATIBILITY must be correctly set for the version of Oracle to which you are connected; otherwise, you will be unable to run any SQL commands. Note that you can set COMPATIBILITY to V7 when connected to Oracle8. This enables you to run Oracle7 SQL against Oracle8.

CON[CAT] {.|c|OFF|ON}

Sets the character you can use to terminate a substitution variable reference if you wish to immediately follow the variable with a character that SQL*Plus would otherwise interpret as a part of the substitution variable name. SQL*Plus resets

the value of CONCAT to a period when you switch
CONCAT on.

COPYC[OMMIT] {<u>0</u>|*n*}

Controls the number of batches after which the
COPY command commits changes to the database.
COPY commits rows to the destination database
each time it copies *n* row batches. Valid values are
zero to 5000. You can set the size of a batch with
the ARRAYSIZE variable. If you set
COPYCOMMIT to zero, COPY performs a commit
only at the end of a copy operation.

COPYTYPECHECK {OFF|<u>ON</u>}

Sets the suppression of the comparison of
datatypes while inserting or appending to tables
with the COPY command. This is to facilitate
copying to DB2, which requires that a CHAR be
copied to a DB2 DATE.

DEF[INE] {<u>&</u>|*c*|OFF|<u>ON</u>}

Sets the character used to prefix substitution
variables to *c*. ON or OFF controls whether
SQL*Plus will scan commands for substitution
variables and replace them with their values. ON
changes the value of *c* back to the default '&', not
the most recently used character. The setting of
DEFINE to OFF overrides the setting of the SCAN
variable. For more information on the SCAN
variable, see the SET SCAN command in
Appendix F.

ECHO {<u>OFF</u>|ON}   Controls whether the START command lists each
command in a command file as the command is
executed. ON lists the commands; OFF suppresses
the listing.

EDITF[ILE] *file_name*[*.ext*]

Sets the default filename for the EDIT command.
For more information about the EDIT command,
see EDIT in this chapter.

You can include a path and/or file extension. For
information on changing the default extension, see
the SUFFIX variable of this command. The default
filename and maximum filename length are
operating system specific.

`EMB[EDDED]` `{`<u>`OFF`</u>`|ON}`

>Controls where on a page each report begins. OFF forces each report to start at the top of a new page. ON allows a report to begin anywhere on a page. Set EMBEDDED to ON when you want a report to begin printing immediately following the end of the previously run report.
>
>**Note:** When you use SET EMBEDDED ON and change the pagesize with SET PAGESIZE *n*, SQL*Plus finishes the current page using the existing pagesize setting and, if required, begins a new page with the new pagesize setting.
>
>**Note:** When you use a BTITLE with SET EMBEDDED ON, the second and subsequent SELECT statements will always begin on a new page. This is because SQL*Plus has no input read ahead. Since SQL*Plus cannot anticipate whether you will enter another SELECT statement or, for example, EXIT, SQL*Plus has to complete processing all output from the first SELECT statement before it reads the next command. This processing includes printing the BTITLE. Therefore, given two SELECT statements, SQL*Plus prints the final BTITLE of the first SELECT statement before it processes the second. The second SELECT statement will then begin at the top of a new page.
>
>**Note:** When you use a REPFOOTER with SET EMBEDDED ON, no footer will be displayed.

`ESC[APE]` `{`$\setminus$`|`*c*`|`<u>`OFF`</u>`|ON}`

>Defines the character you enter as the escape character. OFF undefines the escape character. ON enables the escape character. ON changes the value of *c* back to the default "$\setminus$".
>
>You can use the escape character before the substitution character (set through SET DEFINE) to indicate that SQL*Plus should treat the substitution character as an ordinary character rather than as a request for variable substitution.

`FEED[BACK] {`<u>`6`</u>`|`*`n`*`|OFF|ON}`

Displays the number of records returned by a query when a query selects at least *n* records. ON or OFF turns this display on or off. Turning feedback ON sets *n* to 1. Setting feedback to zero is equivalent to turning it OFF.

`FLAGGER {`<u>`OFF`</u>`|ENTRY|INTERMED[IATE]|FULL}`

Checks to make sure that SQL statements conform to the ANSI/ISO SQL92 standard. If any non-standard constructs are found, the Oracle Server flags them as errors and displays the violating syntax. This is the equivalent of the SQL language ALTER SESSION SET FLAGGER command.

You may execute SET FLAGGER even if you are not connected to a database. FIPS flagging will remain in effect across SQL*Plus sessions until a SET FLAGGER OFF (or ALTER SESSION SET FLAGGER = OFF) command is successful or you exit SQL*Plus.

When FIPS flagging is enabled, SQL*Plus displays a warning for the CONNECT, DISCONNECT, and ALTER SESSION SET FLAGGER commands, even if they are successful.

`FLU[SH] {OFF|`<u>`ON`</u>`}`

Controls when output is sent to the user's display device. OFF allows the host operating system to buffer output. ON disables buffering.

Use OFF only when you run a command file non-interactively (that is, when you do not need to see output and/or prompts until the command file finishes running). The use of FLUSH OFF may improve performance by reducing the amount of program I/O.

`HEA[DING] {OFF|`<u>`ON`</u>`}`

Controls printing of column headings in reports. ON prints column headings in reports; OFF suppresses column headings.

HEADS[EP] {⊥|*c*|OFF|<u>ON</u>}

Defines the character you enter as the heading separator character. The heading separator character cannot be alphanumeric or white space. You can use the heading separator character in the COLUMN command and in the old forms of BTITLE and TTITLE to divide a column heading or title onto more than one line. ON or OFF turns heading separation on or off. When heading separation is OFF, SQL*Plus prints a heading separator character like any other character. ON changes the value of *c* back to the default "|".

LIN[ESIZE] {<u>80</u>|*n*}

Sets the total number of characters that SQL*Plus displays on one line before beginning a new line. It also controls the position of centered and right-aligned text in TTITLE, BTITLE, REPHEADER and REPFOOTER. You can define LINESIZE as a value from 1 to a maximum that is system dependent. Refer to the Oracle installation and user's manual(s) provided for your operating system.

LOBOF[FSET] {*n*|<u>1</u>}

Sets the starting position from which CLOB and NCLOB data is retrieved and displayed.

LONG {<u>80</u>|*n*}

Sets maximum width (in bytes) for displaying LONG, CLOB and NCLOB values; and for copying LONG values. The maximum value of *n* is 2 gigabytes.

When retrieving a LONG value, you may want to retrieve it in increments, however, you must have enough memory to store the entire value set for the LONG column.

LONGC[HUNKSIZE] {<u>80</u>|*n*}

Sets the size (in bytes) of the increments in which SQL*Plus retrieves a LONG, CLOB or NCLOB value.

When retrieving a CLOB or NCLOB value, you may want to retrieve it in increments rather than all at once because of memory size restrictions.

`NEWP[AGE] {`<u>`1`</u>`|`*`n`*`|NONE}`

Sets the number of blank lines to be printed from the top of each page to the top title. A value of zero places a formfeed at the beginning of each page (including the first page) and clears the screen on most terminals. If you set NEWPAGE to NONE, SQL*Plus does not print a blank line or formfeed between the report pages.

`NULL `*`text`*

Sets the text that represents a null value in the result of a SQL SELECT command. Use the NULL clause of the COLUMN command to override the setting of the NULL variable for a given column.

`NUMF[ORMAT] `*`format`*

Sets the default format for displaying numbers. Enter a number format for *format.* For number format descriptions, see the FORMAT clause of the COLUMN command in this chapter.

`NUM[WIDTH] {`<u>`10`</u>`|`*`n`*`}`

Sets the default width for displaying numbers. SQL*Plus rounds numbers up or down to the value of SET NUMWIDTH.

`PAGES[IZE] {`<u>`24`</u>`|`*`n`*`}`

Sets the number of lines in each page. You can set PAGESIZE to zero to suppress all headings, page breaks, titles, the initial blank line, and other formatting information.

`PAU[SE] {`<u>`OFF`</u>`|ON|`*`text`*`}`

Allows you to control scrolling of your terminal when running reports. ON causes SQL*Plus to pause at the beginning of each page of report output. You must press [Return] after each pause. The *text* you enter specifies the text to be displayed each time SQL*Plus pauses. If you enter multiple words, you must enclose *text* in single quotes.

You can embed terminal-dependent escape sequences in the PAUSE command. These sequences allow you to create inverse video messages or other effects on terminals that support such characteristics.

```
RECSEP {WR[APPED]|EA[CH]|OFF}
RECSEPCHAR {_|c}
```
Display or print record separators. A record separator consists of a single line of the RECSEPCHAR (record separating character) repeated LINESIZE times.

RECSEPCHAR defines the record separating character. A single space is the default.

RECSEP tells SQL*Plus where to make the record separation. For example, if you set RECSEP to WRAPPED, SQL*Plus prints a record separator only after wrapped lines. If you set RECSEP to EACH, SQL*Plus prints a record separator following every row. If you set RECSEP to OFF, SQL*Plus does not print a record separator.

```
SERVEROUT[PUT] {OFF|ON} [SIZE n] [FOR[MAT] {WRA[PPED]|
    WOR[D_WRAPPED]]|TRU[NCATED]}]
```
Controls whether to display the output (that is, DBMS_OUTPUT.PUT_LINE) of stored procedures or PL/SQL blocks in SQL*Plus. OFF suppresses the output of DBMS_OUTPUT.PUT_LINE; ON displays the output.

SIZE sets the number of bytes of the output that can be buffered within the Oracle8 Server. The default for *n* is 2000. *n* cannot be less than 2000 or greater than 1,000,000.

When WRAPPED is enabled SQL*Plus wraps the server output within the line size specified by SET LINESIZE, beginning new lines when required.

When WORD_WRAPPED is enabled, each line of server output is wrapped within the line size specified by SET LINESIZE. Lines are broken on word boundaries. SQL*Plus left justifies each line, skipping all leading whitespace.

When TRUNCATED is enabled, each line of server output is truncated to the line size specified by SET LINESIZE.

For each FORMAT, every server output line begins on a new output line.

Note: The output is displayed synchronously after the stored procedure or PL/SQL block has been executed by the Oracle8 Server.

For more information on DBMS_OUTPUT.PUT_LINE, see your *Oracle8 Server Application Developer's Guide.*

SHIFT[INOUT] {VIS[IBLE]|<u>INV[ISIBLE]</u>}

Allows correct alignment for terminals that display shift characters. The SET SHIFTINOUT command is useful for terminals which display shift characters together with data (for example, IBM 3270 terminals). You can only use this command with shift sensitive character sets (for example, JA16DBCS).

Use VISIBLE for terminals that display shift characters as a visible character (for example, a space or a colon). INVISIBLE is the opposite and does not display any shift characters.

SHOW[MODE] {<u>OFF</u>|ON}

Controls whether SQL*Plus lists the *old* and *new* settings of a SQL*Plus system variable when you change the setting with SET. ON lists the settings; OFF suppresses the listing. SHOWMODE ON has the same behavior as the obsolete SHOWMODE BOTH.

SQLC[ASE] {<u>MIX[ED]</u>|LO[WER]|UP[PER]}

Converts the case of SQL commands and PL/SQL blocks just prior to execution. SQL*Plus converts all text within the command, including quoted literals and identifiers, as follows:

- uppercase if SQLCASE equals UPPER

- lowercase if SQLCASE equals LOWER

- unchanged if SQLCASE equals MIXED

SQLCASE does not change the SQL buffer itself.

`SQLCO[NTINUE]` {`≥` |*text*}

> Sets the character sequence SQL*Plus displays as a prompt after you continue a SQL*Plus command on an additional line using a hyphen (–).

`SQLN[UMBER]` {`OFF`|`ON`}

> Sets the prompt for the second and subsequent lines of a SQL command or PL/SQL block. ON sets the prompt to be the line number. OFF sets the prompt to the value of SQLPROMPT.

`SQLPRE[FIX]` {`#`|*c*}

> Sets the SQL*Plus prefix character. While you are entering a SQL command or PL/SQL block, you can enter a SQL*Plus command on a separate line, prefixed by the SQL*Plus prefix character. SQL*Plus will execute the command immediately without affecting the SQL command or PL/SQL block that you are entering. The prefix character must be a non-alphanumeric character.

`SQLP[ROMPT]` {`SQL>`|*text*}

> Sets the SQL*Plus command prompt.

`SQLT[ERMINATOR]` {`;`|*c*|`OFF`|`ON`}

> Sets the character used to end and execute SQL commands to *c*. OFF means that SQL*Plus recognizes no command terminator; you terminate a SQL command by entering an empty line. ON resets the terminator to the default semicolon (;).

`SUF[FIX]` {`SQL`|*text*}

> Sets the default file extension that SQL*Plus uses in commands that refer to command files. SUFFIX does not control extensions for spool files.

`TAB` {`OFF`|`ON`}

> Determines how SQL*Plus formats white space in terminal output. OFF uses spaces to format white space in the output. ON uses the TAB character. TAB settings are every eight characters. The default value for TAB is system dependent.

> **Note:** This option applies only to terminal output. Tabs will not be placed in output files.

`TERM[OUT]` {`OFF`|`ON`}

> Controls the display of output generated by commands executed from a command file. OFF suppresses the display so that you can spool

output from a command file without seeing the output on the screen. ON displays the output. TERMOUT OFF does not affect output from commands you enter interactively.

TI[ME] {<u>OFF</u>|ON}   Controls the display of the current time. ON displays the current time before each command prompt. OFF suppresses the time display.

TIMI[NG] {<u>OFF</u>|ON}

Controls the display of timing statistics. ON displays timing statistics on each SQL command or PL/SQL block run. OFF suppresses timing of each command. For information about the data SET TIMING ON displays, see the Oracle installation and user's manual(s) provided for your operating system. Refer to the TIMING command for information on timing multiple commands.

TRIM[OUT] {OFF|<u>ON</u>}

Determines whether SQL*Plus allows trailing blanks at the end of each displayed line. ON removes blanks at the end of each line, improving performance especially when you access SQL*Plus from a slow communications device. OFF allows SQL*Plus to display trailing blanks. TRIMOUT ON does not affect spooled output.

TRIMS[POOL] {ON|<u>OFF</u>}

Determines whether SQL*Plus allows trailing blanks at the end of each spooled line. ON removes blanks at the end of each line. OFF allows SQL*Plus to include trailing blanks. TRIMSPOOL ON does not affect terminal output.

UND[ERLINE] {<u>–</u>|*c*|<u>ON</u>|OFF}

Sets the character used to underline column headings in SQL*Plus reports to *c*. *c* cannot be an alphanumeric character or a white space. ON or OFF turns underlining on or off. ON changes the value of *c* back to the default "–".

VER[IFY] {OFF|<u>ON</u>}

Controls whether SQL*Plus lists the text of a SQL statement or PL/SQL command before and after SQL*Plus replaces substitution variables with values. ON lists the text; OFF suppresses the listing.

`WRA[P] {OFF`|<u>`ON`</u>`}`	Controls whether SQL*Plus truncates the display of a SELECTed row if it is too long for the current line width. OFF truncates the SELECTed row; ON allows the SELECTed row to wrap to the next line.
	Use the WRAPPED and TRUNCATED clauses of the COLUMN command to override the setting of WRAP for specific columns.

Usage Notes SQL*Plus maintains system variables (also called SET command variables) to allow you to establish a particular environment for a SQL*Plus session. You can change these system variables with the SET command and list them with the SHOW command.

SET ROLE and SET TRANSACTION are SQL commands (see the *Oracle8 Server SQL Reference Manual* for more information). When not followed by the keywords TRANSACTION or ROLE, SET is assumed to be a SQL*Plus command.

Examples The following examples show sample uses of selected SET command variables.

APPINFO To display the setting of APPINFO, enter:

```
SQL> SHOW APPINFO
SQL> appinfo is ON and set to "SQL*Plus"
```

To change the default text, enter:

```
SQL> SET APPI 'This is SQL*Plus'
SQL> SHOW APPINFO
SQL> appinfo is ON and set to "This is SQL*Plus"
```

To make sure that registration has taken place, enter:

```
SQL> VARIABLE MOD VARCHAR2(50)
SQL> VARIABLE ACT VARCHAR2(40)
SQL> EXECUTE DBMS_APPLICATION_INFO.READ_MODULE(:MOD, :ACT);
SQL> PRINT MOD
MOD
--------------------------------------------------
This is SQL*Plus
```

CMDSEP To specify a TTITLE and format a column on the same line:

```
SQL> SET CMDSEP +
SQL> TTITLE LEFT 'SALARIES' + COLUMN SAL FORMAT $9,999
SQL> SELECT ENAME, SAL FROM EMP
  2  WHERE JOB = 'CLERK';
```

The following output results:

```
SALARIES
ENAME            SAL
---------- -------
SMITH            $800
ADAMS          $1,100
JAMES            $950
MILLER         $1,300
```

COLSEP   To set the column separator to "|":

```
SQL> SET COLSEP '|'
SQL> SELECT ENAME, JOB, DEPTNO
  2  FROM EMP
  3  WHERE DEPTNO = 20;
```

The following output results:

```
ENAME      |JOB       |   DEPTNO
-------------------------------
SMITH      |CLERK     |       20
JONES      |MANAGER   |       20
SCOTT      |ANALYST   |       20
ADAMS      |CLERK     |       20
FORD       |ANALYST   |       20
```

COMPATIBILITY   To run a command file, SALARY.SQL, created with Oracle7, enter

```
SQL> SET COMPATIBILITY V7
SQL> START SALARY
```

After running the file, reset compatibility to V8 to run command files created with Oracle8:

```
SQL> SET COMPATIBILITY V8
```

Alternatively, you can add the command SET COMPATIBILITY V7 to the beginning of the command file, and reset COMPATIBILITY to V8 at the end of the file.

ESCAPE   If you define the escape character as an exclamation point (!), then

```
SQL> SET ESCAPE !
SQL> ACCEPT v1 PROMPT 'Enter !&1:'
```

displays this prompt:

```
Enter &1:
```

HEADING   To suppress the display of column headings in a report, enter

```
SQL> SET HEADING OFF
```

If you then run a SQL SELECT command,

```
SQL> SELECT ENAME, SAL FROM EMP
  2  WHERE JOB = 'CLERK';
```

the following output results:

```
ADAMS            1100
JAMES             950
MILLER           1300
```

LOBOFFSET   To set the starting position from which a CLOB column's data is retrieved to the 22nd position, enter

```
SQL> SET LOBOFFSET 22
```

The CLOB data will wrap on your screen; SQL*Plus will not truncate until the 23rd character.

LONG   To set the maximum width for displaying and copying LONG values to 500, enter

```
SQL> SET LONG 500
```

The LONG data will wrap on your screen; SQL*Plus will not truncate until the 501st character.

LONGCHUNKSIZE   To set the size of the increments in which SQL*Plus retrieves LONG values to 100 characters, enter

```
SQL> SET LONGCHUNKSIZE 100
```

The LONG data will be retrieved in increments of 100 characters until the entire value is retrieved or the value of SET LONG is reached.

SERVEROUTPUT   To enable the display of DBMS_OUTPUT.PUT_LINE, enter

```
SQL> SET SERVEROUTPUT ON
```

The following example shows what happens when you execute an anonymous procedure with SET SERVEROUTPUT ON:

```
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE('Task is complete');
  3  END;
  4  /
Task is complete.
```

```
PL/SQL procedure successfully completed.
```

The following example shows what happens when you create a trigger
with SET SERVEROUTPUT ON:

```
SQL> CREATE TRIGGER SERVER_TRIG BEFORE INSERT OR UPDATE -
>    OR DELETE
  2  ON SERVER_TAB
  3  BEGIN
  4  DBMS_OUTPUT.PUT_LINE('Task is complete.');
  5  END;
  6  /
Trigger created.
SQL> INSERT INTO SERVER_TAB VALUES ('TEXT');
Task is complete.
1 row created.
```

To set the output to WORD_WRAPPED, enter

```
SQL> SET SERVEROUTPUT ON FORMAT WORD_WRAPPED
SQL> SET LINESIZE 20
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE('If there is nothing left to do');
  3  DBMS_OUTPUT.PUT_LINE('shall we continue with plan B?');
  4  end;
  5  /
If there is nothing
left to do
shall we continue
with plan B?
```

To set the output to TRUNCATED, enter

```
SQL> SET SERVEROUTPUT ON FORMAT TRUNCATED
SQL> SET LINESIZE 20
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE('If there is nothing left to do');
  3  DBMS_OUTPUT.PUT_LINE('shall we continue with plan B?');
  4  END;
  5  /
If there is nothing
shall we continue wi
```

SHIFTINOUT   To enable the display of shift characters, enter

```
SQL> SET SHIFTINOUT VISIBLE
SQL> SELECT ENAME, JOB FROM EMP;
```

The following output results:

```
ENAME      JOB
---------- ----------
:JJOO:     :AABBCC:
:AA:abc    :DDEE:e
```

where ":" = shift character
        uppercase = multibyte character
        lowercase = singlebyte character

Note: This example illustrates that the columns are aligned correctly. The data used in this example is an illustration only and does not represent real data.

SQLCONTINUE   To set the SQL*Plus command continuation prompt to an exclamation point followed by a space, enter

```
SQL> SET SQLCONTINUE '! '
```

SQL*Plus will prompt for continuation as follows:

```
SQL> TTITLE 'YEARLY INCOME' –
! RIGHT SQL.PNO SKIP 2 –
! CENTER 'PC DIVISION'
SQL>
```

SUFFIX   To set the default command-file extension to UFI, enter

```
SQL> SET SUFFIX UFI
```

If you then enter

```
SQL> GET EXAMPLE
```

SQL*Plus will look for a file named EXAMPLE with an extension of UFI instead of EXAMPLE with an extension of SQL.

SHOW

Purpose   Shows the value of a SQL*Plus system variable or the current SQL*Plus environment.

Syntax   `SHO[W]` *`option`*

where *option* represents one of the following terms or clauses:

```
system_variable
ALL
BTI[TLE]
ERR[ORS] [{FUNCTION|PROCEDURE|PACKAGE|PACKAGE BODY|
   TRIGGER|VIEW|TYPE|TYPE BODY} [schema.]name]
LABEL
LNO
PNO
REL[EASE]
REPF[OOTER]
REPH[EADER]
SPOO[L]
SQLCODE
TTI[TLE]
USER
```

Terms and Clauses   Refer to the following list for a description of each term or clause:

`system_variable`   Represents any system variable set by the SET command.

`ALL`   Lists the settings of all SHOW options, except ERRORS and LABEL, in alphabetical order.

`BTI[TLE]`   Shows the current BTITLE definition.

`ERR[ORS] [{FUNCTION|PROCEDURE|PACKAGE|PACKAGE BODY|TRIGGER|VIEW|TYPE|TYPE BODY} [schema.]name]`

Shows the compilation errors of a stored procedure (includes stored functions, procedures, and packages). After you use the CREATE command to create a stored procedure, a message is displayed if the stored procedure has any compilation errors. To see the errors, you use SHOW ERRORS.

When you specify SHOW ERRORS with no arguments, SQL*Plus shows compilation errors for the most recently created or altered stored procedure. When you specify the type (function, procedure, package, package body, trigger, view, type, or type body) and the name of the PL/SQL stored procedure, SQL*Plus shows errors for that stored procedure. For more information on compilation errors, see your *PL/SQL User's Guide and Reference.*

schema contains the named object. If you omit *schema*, SHOW ERRORS assumes the object is located in your current schema.

Note: You must have DBA privilege to view other schemas, or other schema's object errors.

SHOW ERRORS output displays the line and column number of the error (LINE/COL) as well as the error itself (ERROR). LINE/COL and ERROR have default widths of 8 and 65, respectively. You can alter these widths using the COLUMN command.

LABEL	Shows the security level for the current session. For more information, see your *Trusted Oracle Administrator's Guide.*
LNO	Shows the current line number (the position in the current page of the display and/or spooled output).
PNO	Shows the current page number.
REL[EASE]	Shows the release number of Oracle that SQL*Plus is accessing.
REPF[OOTER]	Shows the current REPFOOTER definition.
REPH[EADER]	Shows the current REPHEADER definition.
SPOO[L]	Shows whether output is being spooled.
SQLCODE	Shows the value of SQL.SQLCODE (the SQL return code of the most recent operation).
TTI[TLE]	Shows the current TTITLE definition.
USER	Shows the username under which you are currently accessing SQL*Plus.

Example   To list the current LINESIZE, enter

```
SQL> SHOW LINESIZE
```

If the current linesize equals 80 characters, SQL*Plus will give the following response:

```
linesize 80
```

The following example illustrates how to create a stored procedure and then show its compilation errors:

```
SQL> connect system/manager
SQL> create procedure scott.proc1 as
SQL> begin
SQL>   :p1 := 1;
SQL> end;
SQL> /
Warning: Procedure created with compilation errors.
SQL> show errors
Errors for PROCEDURE SCOTT.PROC1:
LINE/COL ERROR
--------------------------------------------------------
3/3     PLS-00049: bad bind variable 'P1'
SQL> show errors procedure proc1
No errors.
SQL> show errors procedure scott.proc1
Errors for PROCEDURE SCOTT.PROC1:
LINE/COL ERROR
--------------------------------------------------------
3/3     PLS-00049: bad bind variable 'P1'
```

SPOOL

Purpose   Stores query results in an operating system file and, optionally, sends the file to a printer.

Syntax   `SPO[OL] [file_name[.ext]|OFF|OUT]`

Terms and Clauses   Refer to the following list for a description of each term or clause:

`file_name[.ext]`   Represents the name of the file to which you wish to spool. SPOOL followed by *file_name* begins spooling displayed output to the named file. If you do not specify an extension, SPOOL uses a default extension (LST or LIS on most systems).

`OFF`   Stops spooling.

`OUT`   Stops spooling and sends the file to your host computer's standard (default) printer.

Enter SPOOL with no clauses to list the current spooling status.

Usage Notes   To spool output generated by commands in a command file without displaying the output on the screen, use SET TERMOUT OFF. SET TERMOUT OFF does not affect output from commands run interactively.

Examples   To record your displayed output in a file named DIARY using the default file extension, enter

```
SQL> SPOOL DIARY
```

To stop spooling and print the file on your default printer, type

```
SQL> SPOOL OUT
```

START

Purpose Executes the contents of the specified command file.

Syntax `STA[RT] file_name[.ext] [arg ...]`

Terms and Clauses Refer to the following list for a description of each term or clause:

`file_name[.ext]` Represents the command file you wish to execute. The file can contain any command that you can run interactively.

If you do not specify an extension, SQL*Plus assumes the default command-file extension (normally SQL). For information on changing this default extension, see the SUFFIX variable of the SET command in this chapter.

When you enter START *file_name.ext*, SQL*Plus searches for a file with the filename and extension you specify in the current default directory. If SQL*Plus does not find such a file, SQL*Plus will search a system-dependent path to find the file. Some operating systems may not support the path search. Consult the Oracle installation and user's manual(s) provided for your operating system for specific information related to your operating system environment.

`arg ...` Represent data items you wish to pass to parameters in the command file. If you enter one or more arguments, SQL*Plus substitutes the values into the parameters (&1, &2, and so forth) in the command file. The first argument replaces each occurrence of &1, the second replaces each occurrence of &2, and so forth.

The START command DEFINEs the parameters with the values of the arguments; if you START the command file again in this session, you can enter new arguments or omit the arguments to use the old values.

For more information on using parameters, refer to the subsection "Passing Parameters through the START Command" under "Writing Interactive Commands" in Chapter 3.

Usage Notes   The @ ("at" sign) and @@ (double "at" sign) commands function similarly to START. Disabling the START command in the Product User Profile also disables the @ and @@ commands. See the @ and @@ commands in this chapter for further information on these commands.

The EXIT or QUIT commands in a command file terminate SQL*Plus.

Example   A file named PROMOTE with the extension SQL, used to promote employees, might contain the following command:

```
SELECT * FROM EMP
WHERE MGR=&1 AND JOB='&2' AND SAL>&3;
```

To run this command file, enter

```
SQL> START PROMOTE 7280 CLERK 950
```

SQL*Plus then executes the following command:

```
SELECT * FROM EMP
WHERE MGR=7280 AND JOB='CLERK' AND SAL>950;
```

STORE

Purpose Saves attributes of the current SQL*Plus environment in a host operating system file (a command file).

Syntax
```
STORE {SET} file_name[.ext] [CRE[ATE]|REP[LACE]|
    APP[END]]
```

Terms and Clauses Refer to the following list for a description of each term or clause:

SET Saves the values of the system variables.

Refer to the SAVE command for information on the other terms and clauses in the STORE command syntax.

Usage Notes This command creates a command file which can be executed with the START, @ or @@ commands.

If you want to store a file under a name identical to a STORE command clause (that is, CREATE, REPLACE or APPEND), you must put the name in single quotes or specify a file extension.

Example To store the current SQL*Plus system variables in a file named DEFAULTENV with the default command-file extension, enter

```
SQL> STORE SET DEFAULTENV
```

To append the current SQL*Plus system variables to an existing file called DEFAULTENV with the extension OLD, enter

```
SQL> STORE SET DEFAULTENV.OLD APPEND
```

TIMING

Purpose Records timing data for an elapsed period of time, lists the current timer's name and timing data, or lists the number of active timers.

Syntax `TIMI[NG] [START text|SHOW|STOP]`

Terms and Clauses Refer to the following list for a description of each term or clause:

`START text`   Sets up a timer and makes *text* the name of the timer. You can have more than one active timer by STARTing additional timers before STOPping the first; SQL*Plus nests each new timer within the preceding one. The timer most recently STARTed becomes the current timer.

`SHOW`   Lists the current timer's name and timing data.

`STOP`   Lists the current timer's name and timing data, then deletes the timer. If any other timers are active, the next most recently STARTed timer becomes the current timer.

Enter TIMING with no clauses to list the number of active timers.

Usage Notes You can use this data to do a performance analysis on any commands or blocks run during the period.

For information about the data TIMING displays, see the Oracle installation and user's manual(s) provided for your operating system. Refer to SET TIMING ON for information on automatically displaying timing data after each SQL command or PL/SQL block you run.

To delete all timers, use the CLEAR TIMING command.

Examples To create a timer named SQL_TIMER, enter

```
SQL> TIMING START SQL_TIMER
```

To list the current timer's title and accumulated time, enter

```
SQL> TIMING SHOW
```

To list the current timer's title and accumulated time and to remove the timer, enter

```
SQL> TIMING STOP
```

TTITLE

Purpose Places and formats a specified title at the top of each report page or lists the current TTITLE definition. The old form of TTITLE is used if only a single word or string in quotes follows the TTITLE command.

For a description of the old form of TTITLE, see TTITLE in Appendix F.

Syntax `TTI[TLE] [`*`printspec [text|variable]`* `...]|[`<u>`OFF`</u>`|ON]`

where *printspec* represents one or more of the following clauses used to place and format the *text*:

```
COL n
S[KIP] [n]
TAB n
LE[FT]
CE[NTER]
R[IGHT]
BOLD
FORMAT text
```

Terms and Clauses Refer to the following list for a description of each term or clause. These terms and clauses also apply to the BTITLE command.

text	Represents the title text. Enter *text* in single quotes if you want to place more than one word on a single line.
variable	Represents a user variable or any of the following system-maintained values:

- SQL.LNO (current line number)
- SQL.PNO (current page number)
- SQL.RELEASE (current Oracle release number)
- SQL.SQLCODE (current error code)
- SQL.USER (current username)

To print one of these values, reference the appropriate variable in the title. You can format *variable* with the FORMAT clause.

<u>OFF</u>	Turns the title off (suppresses its display) without affecting its definition.

ON	Turns the title on (restores its display). When you define a top title, SQL*Plus automatically sets TTITLE to ON.
COL *n*	Indents to column *n* of the current line (backward if column *n* has been passed). "Column" in this context means print position, not table column.
S[KIP] [*n*]	Skips to the start of a new line *n* times; if you omit *n,* one time; if you enter zero for *n,* backward to the start of the current line.
TAB *n*	Skips forward *n* columns (backward if you enter a negative value for *n*). "Column" in this context means print position, not table column.
LE[FT], CE[NTER], and R[IGHT]	Left-align, center, and right-align data on the current line respectively. SQL*Plus aligns following data items as a group, up to the end of the *printspec* or the next LEFT, CENTER, RIGHT, or COL command. CENTER and RIGHT use the SET LINESIZE value to calculate the position of the data item that follows.
BOLD	Prints data in bold print. SQL*Plus represents bold print on your terminal by repeating the data on three consecutive lines. On some operating systems, SQL*Plus may instruct your printer to print bolded text on three consecutive lines, instead of bold.
FORMAT *text*	Specifies a format model that determines the format of following data items, up to the next FORMAT clause or the end of the command. The format model must be a *text* constant such as A10 or $999. See COLUMN FORMAT for more information on formatting and valid format models.
	If the datatype of the format model does not match the datatype of a given data item, the FORMAT clause has no effect on that item.
	If no appropriate FORMAT model precedes a given data item, SQL*Plus prints NUMBER values according to the format specified by SET NUMFORMAT or, if you have not used SET NUMFORMAT, the default format. SQL*Plus

prints DATE values according to the default format.

Refer to the FORMAT clause of the COLUMN command in this chapter for more information on default formats.

Enter TTITLE with no clauses to list the current TTITLE definition.

Usage Notes If you do not enter a *printspec* clause before the first occurrence of *text*, TTITLE left justifies the text. SQL*Plus interprets TTITLE in the new form if a valid *printspec* clause (LEFT, SKIP, COL, and so on) immediately follows the command name.

See COLUMN NEW_VALUE for information on printing column and DATE values in the top title.

You can use any number of constants and variables in a *printspec*. SQL*Plus displays the constants and variables in the order you specify them, positioning and formatting each constant or variable as specified by the *printspec* clauses that precede it.

The length of the title you specify with TTITLE cannot exceed 2400 characters.

The continuation character (a hyphen) will not be recognized inside a single-quoted title text string. To be recognized, the continuation character must appear outside the quotes, as follows:

```
SQL> TTITLE CENTER 'Summary Report for' –
> 'the Month of May'
```

Examples To define "Monthly Analysis" as the top title and to left-align it, to center the date, to right-align the page number with a three-digit format, and to display "Data in Thousands" in the center of the next line, enter

```
SQL> TTITLE LEFT 'Monthly Analysis' CENTER '27 Jun 97' –
> RIGHT 'Page:' FORMAT 999 SQL.PNO SKIP CENTER –
> 'Data in Thousands'
```

The following title results:

```
Monthly Analysis                 27 Jun 97           Page:   1
                             Data in Thousands
```

To suppress the top title display without changing its definition, enter

```
SQL> TTITLE OFF
```

UNDEFINE

Purpose   Deletes one or more user variables that you defined either explicitly (with the DEFINE command) or implicitly (with an argument to the START command).

Syntax   `UNDEF[INE]` *variable* `...`

Terms and Clauses   Refer to the following for a description of the term or clause:

variable   Represents the name of the user variable you wish to delete. One or more user variables may be deleted in the same command.

Example   To undefine a user variable named POS, enter

```
SQL> UNDEFINE POS
```

To undefine two user variables named MYVAR1 and MYVAR2, enter

```
SQL> UNDEFINE MYVAR1 MYVAR2
```

VARIABLE

Purpose   Declares a bind variable that can then be referenced in PL/SQL. For more information on bind variables, see "Using Bind Variables" in Chapter 3. For more information about PL/SQL, see your *PL/SQL User's Guide and Reference.*

VARIABLE without arguments displays a list of all the variables declared in the session. VARIABLE followed only by a variable name lists that variable.

Syntax
```
VAR[IABLE] [variable [NUMBER|CHAR|CHAR (n)|NCHAR|
    NCHAR (n)|VARCHAR2 (n)|NVARCHAR2 (n)|CLOB|NCLOB
    REFCURSOR]]
```

Terms and Clauses   Refer to the following list for a description of each term or clause:

variable	Represents the name of the bind variable you wish to create.
NUMBER	Creates a variable of type NUMBER with a fixed length.
CHAR	Creates a variable of type CHAR (character) with a length of one.
CHAR (*n*)	Creates a variable of type CHAR with a maximum length of *n*, up to 2000.
NCHAR	Creates a variable of type NCHAR (national character) with a length of one.
NCHAR (*n*)	Creates a variable of type NCHAR with a maximum length of *n*, up to 2000.
VARCHAR2 (*n*)	Creates a variable of type VARCHAR2 with a maximum length of *n*, up to 4000.
NVARCHAR2 (*n*)	Creates a a variable of type NVARCHAR2 (NCHAR VARYING) with a maximum length of *n*, up to 4000.
CLOB	Creates a variable of type CLOB.
NCLOB	Creates a variable of type NCLOB.
REFCURSOR	Creates a variable of type REF CURSOR.

Usage Notes Bind variables may be used as parameters to stored procedures, or may be directly referenced in anonymous PL/SQL blocks.

To display the value of a bind variable created with VARIABLE, use the PRINT command. For more information, see the PRINT command in this chapter.

To automatically display the value of a bind variable created with VARIABLE, use the SET AUTOPRINT command. For more information, see the SET AUTOPRINT command in this chapter.

Bind variables cannot be used in the COPY command or SQL statements, except in PL/SQL blocks. Instead, use substitution variables.

When you execute a VARIABLE ... CLOB or NCLOB command, SQL*Plus associates a LOB locator with the bind variable. The LOB locator is automatically populated when you execute a SELECT clob_column INTO :cv statement in a PL/SQL block. SQL*Plus closes the LOB locator after completing a PRINT statement for that bind variable, or when you exit SQL*Plus.

SQL*Plus SET commands such as SET LONG and SET LONGCHUNKSIZE and SET LOBOFFSET may be used to control the size of the buffer while PRINTing CLOB or NCLOB bind variables.

SQL*Plus REFCURSOR bind variables may be used to reference PL/SQL 2.3 or higher Cursor Variables, allowing PL/SQL output to be formatted by SQL*Plus. For more information on PL/SQL Cursor Variables, see your *PL/SQL User's Guide and Reference.*

When you execute a VARIABLE ... REFCURSOR command, SQL*Plus creates a cursor bind variable. The cursor is automatically opened by an OPEN ... FOR SELECT statement referencing the bind variable in a PL/SQL block. SQL*Plus closes the cursor after completing a PRINT statement for that bind variable, or on exit.

SQL*Plus formatting commands such as BREAK, COLUMN, COMPUTE and SET may be used to format the output from PRINTing a REFCURSOR.

A REFCURSOR bind variable may not be PRINTed more than once without re-executing the PL/SQL OPEN ... FOR statement.

Examples   The following example illustrates creating a bind variable and then setting it to the value returned by a function:

```
SQL> VARIABLE id NUMBER
SQL> BEGIN
  2    :id := emp_management.hire
  3      ('BLAKE','MANAGER','KING',2990,'SALES');
  4  END;
```

The bind variable named `id` can be displayed with the PRINT command or used in subsequent PL/SQL subprograms.

The following example illustrates automatically displaying a bind variable:

```
SQL> SET AUTOPRINT ON
SQL> VARIABLE a REFCURSOR
SQL> BEGIN
  2  OPEN :a FOR SELECT * FROM DEPT ORDER BY DEPTNO;
  3  END;
  4  /

PL/SQL procedure successfully completed.

DEPTNO   DNAME         LOC
-------- ------------- -------------
      10 ACCOUNTING    NEW YORK
      20 RESEARCH      DALLAS
      30 SALES         CHICAGO
      40 OPERATIONS    BOSTON
```

In the above example, there is no need to issue a PRINT command to display the variable.

The following example creates some variables and then lists them:

```
SQL> VARIABLE id NUMBER
SQL> VARIABLE txt CHAR (20)
SQL> VARIABLE myvar REFCURSOR
SQL> VARIABLE
variable id
datatype NUMBER

variable txt
datatype CHAR(20)

variable myvar
datatype REFCURSOR
```

The following example lists a single variable:

```
SQL> VARIABLE txt
variable txt
datatype CHAR(20)
```

The following example illustrates producing a report listing individual salaries and computing the departmental and total salary cost:

```
SQL> VARIABLE RC REFCURSOR
  2  BEGIN
  3    OPEN :RC FOR SELECT DNAME, ENAME, SAL
  4             FROM EMP, DEPT
  5             WHERE EMP.DEPTNO = DEPT.DEPTNO
  6             ORDER BY EMP.DEPTNO, ENAME;
  7  END;
  8  /

PL/SQL procedure successfully completed.

SQL> SET PAGESIZE 100 FEEDBACK OFF
SQL> TTITLE LEFT '*** Departmental Salary Bill ***' SKIP 2
SQL> COLUMN SAL FORMAT $999,990.99 HEADING 'Salary'
SQL> COLUMN DNAME HEADING 'Department'
SQL> COLUMN ENAME HEADING 'Employee'
SQL> COMPUTE SUM LABEL 'Subtotal:' OF SAL ON DNAME
SQL> COMPUTE SUM LABEL 'Total:' OF SAL ON REPORT
SQL> BREAK ON DNAME SKIP 1 ON REPORT SKIP 1
SQL> PRINT RC

*** Departmental Salary Bill ***


Department      Employee     Salary
--------------  ------------ ----------
ACCOUNTING      CLARK          $2,450.00
                KING           $5,000.00
                MILLER         $1,300.00
**************               ----------
Subtotal:                      $8,750.00

RESEARCH        ADAMS          $1,100.00
                FORD           $3,000.00
                JONES          $2,975.00
                SCOTT          $3,000.00
```

```
              SMITH          $800.00
**************           ----------
Subtotal:                $10,875.00

SALES         ALLEN        $1,600.00
              BLAKE        $2,850.00
              JAMES          $950.00
              MARTIN       $1,250.00
              TURNER       $1,500.00
              WARD         $1,250.00
**************           ----------
Subtotal:                 $9,400.00


                         ----------
Total:                   $29,025.00
```

The following example illustrates producing a report containing a CLOB column, and then displaying it with the SET LOBOFFSET command.

Assume you have already created a table named clob_tab which contains a column named clob_col of type CLOB. The clob_col contains the following data:

```
Remember to run the Departmental Salary Bill report each
month. This report contains confidential information.
```

To produce a report listing the data in the col_clob column, enter

```
SQL> variable t clob
SQL> begin
  2    select clob_col into t: from clob_tab;
  3  end;
  4  /
PL/SQL procedure successfully completed
```

To print 200 characters from the column clob_col, enter

```
SQL> set LONG 200
SQL> print t
```

The following output results:

```
T
------------------------------------------------------------
Remember to run the Departmental Salary Bill report each
month. This report contains confidential information.
```

To set the printing position to the 21st character, enter

```
SQL> set LOBOFFSET 21
SQL> print t
```

the following output results:

```
T
------------------------------------------------------------
Departmental Salary Bill report each month. This report
contains confidential information.
```

For more information on creating CLOB columns, see your *Oracle8 Server SQL Reference Manual.*

WHENEVER OSERROR

Purpose   Exits SQL*Plus if an operating system error occurs (such as a file I/O error).

Syntax
```
WHENEVER OSERROR
    {EXIT [SUCCESS|FAILURE|n|variable|:BindVariable]
    [COMMIT|ROLLBACK]|CONTINUE [COMMIT|ROLLBACK|NONE]}
```

Terms and Clauses   Refer to the following list for a description of each term or clause:

`EXIT [SUCCESS|FAILURE|n|variable|:BindVariable]`
    Directs SQL*Plus to exit as soon as an operating system error is detected. You can also specify that SQL*Plus return a success or failure code, the operating system failure code, or a number or variable of your choice. See EXIT in this chapter for details.

`CONTINUE`   Turns off the EXIT option.

`COMMIT`   Directs SQL*Plus to execute a COMMIT before exiting or continuing and save pending changes to the database.

`ROLLBACK`   Directs SQL*Plus to execute a ROLLBACK before exiting or continuing and abandon pending changes to the database.

`NONE`   Directs SQL*Plus to take no action before continuing.

Usage Notes   If you do not enter the WHENEVER OSERROR command, the default behavior of SQL*Plus is to continue and take no action when an operating system error occurs.

Examples   The commands in the following command file cause SQL*Plus to exit and COMMIT any pending changes if a failure occurs when writing to the output file:

```
WHENEVER OSERROR EXIT SQL.OSCODE COMMIT
SPOOL MYLOG
UPDATE EMP SET SAL = SAL*1.1
COPY TO SCOTT/TIGER@HQDB –
REPLACE EMP –
USING SELECT * FROM EMP
SPOOL OUT
SELECT SAL FROM EMP
```

WHENEVER SQLERROR

Purpose   Exits SQL*Plus if a SQL command or PL/SQL block generates an error.

Syntax
```
WHENEVER SQLERROR
    {EXIT [SUCCESS|FAILURE|WARNING|n|variable|:BindVariable]
    [COMMIT|ROLLBACK]|CONTINUE [COMMIT|ROLLBACK|NONE]}
```

Terms and Clauses   Refer to the following list for a description of each term or clause:

`EXIT [SUCCESS|FAILURE|WARNING|n|variable|:BindVariable]`

Directs SQL*Plus to exit as soon as it detects a SQL command or PL/SQL block error (but after printing the error message). SQL*Plus will not exit on a SQL*Plus error. The EXIT clause of WHENEVER SQLERROR follows the same syntax as the EXIT command. See EXIT in this chapter for details.

`CONTINUE`   Turns off the EXIT option.

`COMMIT`   Directs SQL*Plus to execute a COMMIT before exiting or continuing and save pending changes to the database.

`ROLLBACK`   Directs SQL*Plus to execute a ROLLBACK before exiting or continuing and abandon pending changes to the database.

`NONE`   Directs SQL*Plus to take no action before continuing.

Usage Notes   The WHENEVER SQLERROR command is triggered by SQL command or PL/SQL block errors, and not by SQL*Plus command errors.

If you do not enter the WHENEVER SQLERROR command, the default behavior of SQL*Plus is to continue and take no action when a SQL error occurs.

Examples   The commands in the following command file cause SQL*Plus to exit and return the SQL error code if the SQL UPDATE command fails:

```
SQL> WHENEVER SQLERROR EXIT SQL.SQLCODE
SQL> UPDATE EMP SET SAL = SAL*1.1
```

The following SQL command error causes SQL*Plus to exit and return the SQL error code:

```
SQL> WHENEVER SQLERROR EXIT SQL.SQLCODE
SQL> SELECT COLUMN_DOES_NOT_EXIST FROM DUAL;
SELECT COLUMN_DOES_NOT_EXIST FROM DUAL
       *
ERROR at line 1:
ORA-00904: invalid column name

Disconnected from Oracle.....
```

The following SQL command error causes SQL*Plus to exit and return
the value of the variable MY_ERROR_VAR:

```
SQL> DEFINE MY_ERROR_VAR = 99
SQL> WHENEVER SQLERROR EXIT MY_ERROR_VAR
SQL> UPDATE NON_EXISTED_TABLE SET COL1 = COL1 + 1;

UPDATE NON_EXISTED_TABLE SET COL1 = COL1 + 1
       *
ERROR at line 1:
ORA-00942: table or view does not exist

Disconnected from Oracle.....
```

The following examples show that the WHENEVER SQLERROR
command does not have any effect on SQL*Plus commands, but does
on SQL commands and PL/SQL blocks:

```
SQL> WHENEVER SQLERROR EXIT SQL.SQLCODE
SQL> COLUMN ENAME HEADIING "EMPLOYEE NAME"

Unknown COLUMN option "HEADIING"

SQL> SHOW NON_EXISTED_OPTION

Unknown SHOW option "NON_EXISTED_OPTION"

SQL> GET NON_EXISTED_FILE.SQL

Unable to open "NON_EXISTED_FILE.SQL"

SQL>
```

The following PL/SQL block error causes SQL*Plus to exit and return
the SQL error code:

```
SQL> WHENEVER SQLERROR EXIT SQL.SQLCODE
SQL> BEGIN
  2    SELECT COLUMN_DOES_NOT_EXIST FROM DUAL;
  3  END;
  4  /

SELECT COLUMN_DOES_NOT_EXIST FROM DUAL;
       *
ERROR at line 2:
ORA-06550: line 2, column 10:
PLS-00201: identifier 'COLUMN_DOES_NOT_EXIST' must be
declared
ORA-06550: line 2, column 3:
PL/SQL: SQL Statement ignored

Disconnected from Oracle.....
```

APPENDIX

A

T

CPY0002:

Cause:

Action:

CPY0003:

Cause:

Action:

A-2 SQL*Plus User's Guide and Reference

Release 8.0 Enhancements

S QL*Plus Release 8.0 provides a number of enhancements over previous releases of SQL*Plus. This appendix describes the enhancements for SQL*Plus Release 8.0.

SQL*Plus Release 8.0 Enhancements

SQL*Plus Release 8.0 is a superset of SQL*Plus Release 3.3.

To fully exploit SQL*Plus Release 8.0, you need Oracle8. SQL*Plus Release 8.0 gives you the following capabilities:

- There is a new command named ATTRIBUTE. The ATTRIBUTE command displays attributes for a given column, and functions similar to the COLUMN command.

- There is a new command named PASSWORD. The PASSWORD command allows passwords to be changed without echoing the password to an input device.

- The CONNECT command will now prompt a user to change their password if it has expired.

- The EXIT command now has a :BindVariable clause. The :BindVariable clause represents a variable created in SQL*Plus with the VARIABLE command, and then referenced in PL/SQL or other subprograms. :BindVariable exits the subprogram and returns you to SQL*Plus.

- The LONG and LONGCHUNKSIZE datatypes determine the limits for the CLOB and NCLOB datatypes.

- The SET command now has a LOBOFFSET clause. The LOBOFFSET clause sets the starting position from which CLOB and NCLOB data is retrieved.

- The SET NEWPAGE command now has a NONE clause. A value of NONE, prints no blank lines and no formfeed between report pages.

- The SET command now has a SHIFTINOUT clause. The SHIFTINOUT clause allows correct alignment for terminals that display shift characters. This command can only be used with shift sensitive character sets.

- The SHOW ERRORS command now includes the TYPE and TYPE BODY clauses.

- The VARIABLE command now includes the clauses NCHAR, NVARCHAR2 (NCHAR VARYING), CLOB and NCLOB.

- The maximum length of CHAR and NCHAR bind variables have been increased to 2000.

- The maximum length of VARCHAR2 and NVARCHAR2 have been increased to 4000.

SQL*Plus Limits

Table C–1, on the following page, lists the limit, or maximum value, of each of the SQL*Plus elements shown. The limits shown are valid for most operating systems.

Item	Limit
filename length	system dependent
username length	30 bytes
user variable name length	30 bytes
user variable value length	240 characters
command-line length	2500 characters
length of a LONG value entered through SQL*Plus	LINESIZE value
LINESIZE	system dependent
LONGCHUNKSIZE value	system dependent
output line size	system dependent
line size after variable substitution	3,000 characters (internal only)
number of characters in a COMPUTE command label	500 characters
number of lines per SQL command	500 (assuming 80 characters per line)
maximum PAGESIZE	50,000 lines
total row width	60,000 characters for VMS; otherwise, 32,767 characters
maximum ARRAYSIZE	5000 rows
maximum number of nested command files	20 for VMS, CMS, Unix; otherwise, 5
maximum page number	99,999
maximum PL/SQL error message size	2K
maximum ACCEPT character string length	240 Bytes

Table C – 1

D

SQL Command List

Table D–1, on the following page, lists major SQL commands. Refer to the *Oracle8 Server SQL Reference Manual* for full documentation of these commands.

Major SQL Commands and Clauses

ALTER	LOCK TABLE
ANALYZE	NOAUDIT
AUDIT	RENAME
COMMENT	REVOKE
COMMIT	ROLLBACK
CREATE	SAVEPOINT
DELETE	SELECT
DROP	SET ROLE
EXPLAIN	SET TRANSACTION
GRANT	TRUNCATE
INSERT	UPDATE

Table D – 1 SQL Command List

E Security

This appendix describes the available methods for controlling access to database tables and SQL*Plus commands. The available methods for security fall into two broad categories:

- SQL*Plus PRODUCT_USER_PROFILE table

- roles

PRODUCT_USER_PROFILE Table

Various Oracle products use PRODUCT_USER_PROFILE, a table in the SYSTEM account, to provide product-level security that supplements the user-level security provided by the SQL GRANT and REVOKE commands and user roles.

Overview
DBAs can use PRODUCT_USER_PROFILE to disable certain SQL and SQL*Plus commands in the SQL*Plus environment on a per-user basis. SQL*Plus—not Oracle—enforces this security. DBAs can even restrict access to the GRANT, REVOKE, and SET ROLE commands to control users' ability to change their database privileges.

SQL*Plus reads restrictions from PRODUCT_USER_PROFILE when a user logs in to SQL*Plus and maintains those restrictions for the duration of the session. Changes to PRODUCT_USER_PROFILE will only take effect the next time the affected users log in to SQL*Plus.

Creating the Table
You can create PRODUCT_USER_PROFILE by running the command file named PUPBLD with the extension SQL as SYSTEM. The exact format of the file extension and the location of the file are system dependent. See the Oracle installation and user's manual(s) provided for your operating system or your DBA for more information.

> **Note:** If the table is created incorrectly, all users other than SYSTEM will see a warning when connecting to Oracle that the PRODUCT_USER_PROFILE information is not loaded.

Table Structure
The PRODUCT_USER_PROFILE table consists of the following columns:

```
PRODUCT                          NOT NULL CHAR (30)
USERID                                    CHAR(30)
ATTRIBUTE                                 CHAR(240)
SCOPE                                     CHAR(240)
NUMERIC_VALUE                             NUMBER(15,2)
CHAR_VALUE                                CHAR(240)
DATE_VALUE                                DATE
LONG_VALUE                                LONG
```

Description and Use of Columns Refer to the following list for the descriptions and use of each column in the PRODUCT_USER_PROFILE table:

Product   Must contain the product name (in this case "SQL*Plus"). You cannot enter wildcards or NULL in this column. Also notice that the product name SQL*Plus must be specified in mixed case, as shown, in order to be recognized.

Userid   Must contain the username (in uppercase) of the user for whom you wish to disable the command. To disable the command for more than one user, use SQL wild cards (%) or make multiple entries. Thus, all of the following entries are valid:

- SCOTT

- CLASS1

- CLASS% (all users whose names start with CLASS)

- % (all users)

Attribute   Must contain the name (in uppercase) of the SQL, SQL*Plus, or PL/SQL command you wish to disable (for example, GET). If you are disabling a role, it must contain the character string "ROLES". You cannot enter a wildcard. See "Administration" below for a list of SQL and SQL*Plus commands you can disable. See "Roles" below for information on how to disable a role.

Scope   SQL*Plus ignores this column. It is recommended that you enter NULL in this column. Other products may store specific file restrictions or other data in this column.

Numeric_Value   SQL*Plus ignores this column. It is recommended that you enter NULL in this column. Other products may store numeric values in this column.

Char_Value   Must contain the character string "DISABLED" to disable a SQL, SQL*Plus, or PL/SQL command. If you are disabling a role, it must contain the name of the role you wish to disable. You cannot use a wildcard. See "Roles" below for information on how to disable a role.

Date_Value	SQL*Plus ignores this column. It is recommended that you enter NULL in this column. Other products may store DATE values in this column.	
Long_Value	SQL*Plus ignores this column. It is recommended that you enter NULL in this column. Other products may store LONG values in this column.	

Administration

The DBA username SYSTEM owns and has all privileges on PRODUCT_USER_PROFILE. (When SYSTEM logs in, SQL*Plus does not read PRODUCT_USER_PROFILE. Therefore, no restrictions apply to user SYSTEM.) Other Oracle usernames should have only SELECT access to this table, which allows a view of restrictions of that username and those restrictions assigned to PUBLIC. The command file PUPBLD, when run, grants SELECT access on PRODUCT_USER_PROFILE to PUBLIC.

Disabling SQL*Plus, SQL, and PL/SQL Commands

To disable a SQL or SQL*Plus command for a given user, insert a row containing the user's username in the Userid column, the command name in the Attribute column, and DISABLED in the Char_Value column.

The Scope, Numeric_Value, and Date_Value columns should contain NULL. For example:

```
                                      NUMERIC  CHAR     DATE
PRODUCT   USERID  ATTRIBUTE  SCOPE    VALUE    VALUE    VALUE
--------  ------  ---------  -----    -------  -------  -------
SQL*Plus  SCOTT   HOST                         DISABLED

SQL*Plus  %       INSERT                       DISABLED

SQL*Plus  %       UPDATE                       DISABLED

SQL*Plus  %       DELETE                       DISABLED
```

To re-enable commands, delete the row containing the restriction.

You can disable the following SQL*Plus commands:

- COPY
- EDIT
- EXECUTE
- EXIT
- GET
- HOST (or your operating system's alias for HOST, such as $ on VMS and ! on UNIX)

- QUIT

- PASSWORD

- RUN

- SAVE

- SET (see note below)

- SPOOL

- START

   Note: Disabling the SQL*Plus SET command will also disable
   the SQL SET ROLE and SET TRANSACTION commands.
   Disabling the SQL*Plus START command will also disable the
   SQL*Plus @ and @@ commands.

You can also disable the following SQL commands:

- ALTER

- ANALYZE

- AUDIT

- CONNECT

- CREATE

- DELETE

- DROP

- GRANT

- INSERT

- LOCK

- NOAUDIT

- RENAME

- REVOKE

- SELECT

- SET ROLE

- SET TRANSACTION

- TRUNCATE

- UPDATE

You can also disable the following PL/SQL commands:

- BEGIN
- DECLARE

    Note: Disabling BEGIN and DECLARE does not prevent the use of the SQL*Plus EXECUTE command. EXECUTE must be disabled separately.

Disabling SET ROLE   From SQL*Plus, users can submit any SQL command. In certain situations, this can cause security problems. Unless you take proper precautions, a user could use SET ROLE to access privileges obtained via an application role. With these privileges, they might issue SQL statements from SQL*Plus that could wrongly change database tables.

To prevent application users from accessing application roles in SQL*Plus, you can use PRODUCT_USER_PROFILE to disable the SET ROLE command. This allows a SQL*Plus user only those privileges associated with the roles enabled when they started SQL*Plus. For more information about the creation and usage of user roles, see your *Oracle8 Server SQL Reference* and *Oracle8 Server Administrator's Guide.*

Disabling Roles   To disable a role for a given user, insert a row in PRODUCT_USER_PROFILE containing the user's username in the Userid column, "ROLES" in the Attribute column, and the role name in the Char_Value column.

    Note: When you enter "PUBLIC" or "%" for the Userid column, you disable the role for all users. You should only use "%" or "PUBLIC" for roles which are granted to "PUBLIC". If you try to disable a role that has not been granted to a user, none of the roles for that user are disabled.

The Scope, Numeric_Value, and Date_Value columns should contain NULL. For example:

```
                                      NUMERIC  CHAR     DATE
PRODUCT   USERID  ATTRIBUTE  SCOPE  VALUE    VALUE    VALUE
--------  ------  ---------  -----  -------  -------  -------
SQL*Plus  SCOTT   ROLES                      ROLE1
SQL*Plus  PUBLIC  ROLES                      ROLE2
```

During login, these table rows are translated into the command

```
SET ROLE ALL EXCEPT ROLE1, ROLE2
```

To ensure that the user does not use the SET ROLE command to change their roles after login, you can disable the SET ROLE command. See "Disabling SET ROLE" earlier in this appendix.

To re-enable roles, delete the row containing the restriction.

Roles

To provide for the security of your database tables in Oracle8 using SQL commands, you can create and control access to roles. By creating a role and then controlling who has access to it, you can ensure that only certain users have access to particular database privileges.

Overview

Roles are created and used with the SQL CREATE, GRANT, and SET commands:

- To create a role, you use the CREATE command. You can create roles with or without passwords.

- To grant access to roles, you use the GRANT command. In this way, you can control who has access to the privileges associated with the role.

- To access roles, you use the SET ROLE command. If you created the role with a password, the user must know the password in order to access the role.

For more information about roles, see your *Oracle8 Server SQL Reference*, your *Oracle8 Server Administrator's Guide*, and your *Oracle8 Server Concepts Manual.*

APPENDIX

F

T

BTITLE (old form)

Purpose

Syntax

Notes

COLUMN DEFAULT

Purpose

Syntax

Notes

DOCUMENT

Purpose

Syntax

Notes

NEWPAGE

SET BUFFER

SET DOCUMENT

SQL Commands 5–3

SET SCAN

Purpose

Syntax SET SCAN OFF | ON

Notes

SET SPACE

Syntax SET SPACE {1 | n}

Notes

SET TRUNCATE

Purpose

Syntax SET TRUNCATE OFF | ON

Notes

Glossary

Glossary–5

Glossary–7

Glossary–9

Glossary SQL*Plus Reference

Index

. (period), 2–9
; (semicolon), 2–6
: (colon), bind variables, 3–27
:BindVariable clause, EXIT command, 7–59
& (ampersand), substitution variables, 3–18
# Pound sign, 7–29
$, number format, 4–4
@ ("at" sign)
  in CONNECT command, 5–3, 7–41
  in COPY command, 5–4, 7–43
  in SQLPLUS command, 3–13, 5–3, 6–2
@ ("at" sign) command, 3–13, 3–17, 7–5
  arguments, 7–5
  command file, 3–13, 7–5
  passing parameters to a command file, 7–5
  similar to START, 3–13, 7–6, 7–104
@@ (double "at" sign) command, 3–17, 7–7
  command file, 7–7
  similar to START, 7–7, 7–104
– (hyphen), continuing a long SQL*Plus
     command, 2–11, 7–1
– (hyphen) clause, 6–3
–? clause, 6–3
–– (comment delimiter), 3–11
–~ Negative infinity sign, 7–29
–SILENT clause, 6–2
* (asterisk)
  in DEL command, 3–2, 7–48
  in LIST command, 3–3, 7–66

/ (slash) command, 7–9
  entered at buffer line–number prompt, 2–7,
    7–9
  entered at command prompt, 2–9, 7–9
  executing current PL/SQL block, 2–10
  executing current SQL command, 2–9
  similar to RUN, 2–9, 7–9, 7–77
/ (slash), default logon, 6–3, 7–41
/*...*/ (comment delimiters), 3–11
/NOLOG option, 6–3
~ Infinity sign, 7–29
_EDITOR, in EDIT command, 3–7, 7–56
0, number format, 4–4
9, number format, 4–4

A

ACCEPT command, 3–25, 7–10
  and DEFINE command, 7–46
  CHAR clause, 7–10
  customizing prompts for value, 3–26
  DATE clause, 7–10
  DEFAULT clause, 7–10
  FORMAT clause, 7–10
  HIDE clause, 7–11
  NOPROMPT clause, 7–11
  NUMBER clause, 3–26
  PROMPT clause, 3–25, 7–10
Access, denying and granting, E–2

SQLPROMPT variable, 7–92

SQLTERMINATOR variable, 7–63, 7–92

START clause, 7–106

START command, 3–12, 7–103
  arguments, 3–23, 7–103
  command file, 3–12, 7–103
  disabling, E–5
  passing parameters to a command file, 3–23,
    7–103
  similar to @ ("at" sign) command, 3–13, 7–6,
    7–104
  similar to @@ (double "at" sign) command,
    7–7, 7–104

Starting SQL*Plus, 2–3
  shortcuts, 2–4

Statistics, 3–32

STD function, 4–15

STOP clause, 7–106

STORE command, 3–16, 7–105
  SET clause, 7–105

Stored functions, 3–29

Stored procedures, creating, 2–8

Substitution variables, 3–18
  appending characters immediately after,
    3–20
  avoiding unnecessary prompts for value,
    3–20
  concatenation character, 7–84
  DEFINE command, 3–21, 7–46
  prefixing, 7–85
  restrictions, 3–22
  single and double ampersands, 3–20
  system variables used with, 3–22
  undefined, 3–19
  where and how to use, 3–19

SUCCESS clause, 7–59

SUFFIX variable, 7–92
  used with @ ("at" sign) command, 7–5
  used with @@ (double "at" sign) command,
    7–7
  used with EDIT command, 7–56
  used with GET command, 7–61
  used with SAVE command, 7–78
  used with START command, 7–103

SUM function, 4–15

Summary lines
  computing and printing, 4–14, 7–35
  computing and printing at ends of reports,
    4–18
  computing same type on different columns,
    4–19
  printing "grand" and "sub" summaries
    (totals), 4–19
  printing multiple on same break column,
    4–20

Syntax
  conventions, 1–4
  COPY command, 5–4

Syntax rules
  SQL commands, 2–7
  SQL*Plus commands, 2–11

SYSDATE, 4–30

System variables, 2–12, 7–94
  changing current settings, 7–79
  listing current settings, 2–12, 7–99
  listing old and new values, 7–91
  storing and restoring, 3–16
  used with substitution variables, 3–22

System–maintained values
  displaying in headers and footers, 7–74
  displaying in titles, 4–26, 7–107
  formatting in titles, 4–27

T

TAB clause, 7–75, 7–108

TAB variable, 7–92

Tables, 1–2
  access to sample, 1–8
  controlling destination when copying, 5–6,
    7–44
  copying values between, 5–4, 5–8, 7–43
  DEPT, 1–6
  EMP, 1–6
  listing column definitions, 2–14, 7–50
  referring to another user's when copying,
    5–8
  sample, 1–6

Reader's Comment Form

Name of Document: SQL*Plus® User's Guide and Reference
Part No. A53717–01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?

- Is the information clearly presented?

- Do you need more information? If so, where?

- Are the examples correct? Do you need more examples?

- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

_____

_____

_____

_____

_____

_____

_____

Please send or email your comments to:

> SQL*Plus Documentation Manager
> Oracle Systems Australia Pty Ltd
> 324 St. Kilda Road
> Melbourne VIC 3004 Australia
> +61 3 9209 1600   +61 3 9690 0043 (fax)
> sqlplus@au.oracle.com

If you would like a reply, please give your name, address, and telephone number below:

_____

_____

_____

Thank you for helping us improve our documentation.