SQL*Plus®

User's Guide and Reference

23ai
F47057-05
May 2024

ORACLE®

SQL*Plus User's Guide and Reference, 23ai

F47057-05

Copyright © 1996, 2024, Oracle and/or its affiliates.

Primary Author: Gunjan Jain

Contributors: Luan Nim, Andrei Souleimanian, Senthilprabhu Dhamotharan, Mahantesh Savanur

Contents

2 Configuring SQL*Plus

3 Starting SQL*Plus

Part II   Using SQL*Plus

4   SQL*Plus Basics

5   Using Scripts in SQL*Plus

6   Using Substitution Variables

7   Formatting SQL*Plus Reports

8   Generating Reports from SQL*Plus

9   Tuning SQL*Plus

10   SQL*Plus Security

11   Database Administration with SQL*Plus

12   SQL*Plus Globalization Support

Part III   SQL*Plus Reference

13   SQL*Plus Command Reference

Part IV SQL*Plus Appendixes

A SQL*Plus Limits

B SQL*Plus COPY Command

C   Obsolete SQL*Plus Commands

D   SQL*Plus Instant Client

Index

Preface

The SQL*Plus (pronounced "sequel plus") User's Guide and Reference introduces SQL*Plus and its uses, and describes each SQL*Plus command.

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

The *SQL*Plus User's Guide and Reference* is intended for business and technical users and system administrators who perform the following tasks:

- Develop and run batch scripts
- Format, calculate on, store, print and create web output from query results
- Examine table and object definitions
- Perform database administration

This document assumes a basic understanding of the SQL language. If you do not have familiarity with SQL, see the *Oracle Database SQL Language Reference*. If you plan to use PL/SQL with SQL*Plus, see the *Oracle Database PL/SQL Language Reference*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database PL/SQL Language Reference*
- *Oracle Database SQL Language Reference*

- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Database Backup and Recovery User's Guide*
- *Oracle Database Advanced Application Developer's Guide*
- *Oracle XML DB Developer's Guide*
- *Oracle Database Globalization Support Guide*
- *Oracle Database Heterogeneous Connectivity Administrator's Guide*
- *Oracle Database Error Messages*
- *Oracle Database Upgrade Guide*
- *Oracle Database Reference*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database Net Services Administrator's Guide*
- *Pro*COBOL Programmer's Guide*
- *Pro*C/C++ Programmer's Guide*
- Oracle Database installation and user's manuals for your operating system

Many of the examples in this book use the sample schemas, which you need to download from the GitHub repository. See *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Database SQL*Plus User's Guide and Reference

This preface contains the changes in this book for Oracle Database 23ai.

New Features

This section lists new features introduced in SQL*Plus releases.

SQL*Plus Release 23ai

This section describes new features introduced in SQL*Plus in 23ai.

- A new command, `CONFIG`, generates the config store JSON file from the `tnsnames.ora` file.

  CONFIG

- A new command, `SHOW CONNECTION`, lists the Oracle Net Service Names present in the `tnsnames.ora` file and resolves the given net service name to the connection string.

  SHOW CONNECTION

- A new command, `ARGUMENT`, customizes the input prompt and sets default values when parameters are not passed to the script.

  ARGUMENT

- A new command, `PING`, pings the database or database network listener to check availability.

  PING

- Display error details

  - A new command, `OERR`, allows users to see the detailed cause and action text for Oracle errors.
    OERR

  - A new command, `SET ERRORDETAILS`, displays the Oracle Error Help URL along with the cause and action details when any SQL, PL/SQL, or SQL*Plus statement fails during execution.
    SET ERRORDETAILS

  - The `HELP` command is enhanced to display error details, such as Cause, Action, and Parameters, for the provided error code.
    HELP

- The `DESCRIBE` command is enhanced to display annotation information for columns that have associated annotations available.

DESCRIBE

- Support for the `SQL BOOLEAN` data type

  – Added support for a new data type, `BOOLEAN`, in SQL statements.
    DESCRIBE

  – Enhanced the syntax of the `COLUMN` and the `VARIABLE` commands.
    COLUMN

    VARIABLE

- Support for a new data type, `VECTOR`, in SQL*Plus.
  VARIABLE

SQL*Plus Release 21c

This section describes new features introduced in SQL*Plus in 21c.

- The `HISTORY` command syntax has been enhanced. If *N* is omitted in the command, then `RUN`, `EDIT` or `DELETE` is executed on the most recent entry in the history list.

- `SET JSONPRINT` is a new command that enables you to format JSON data.

SQL*Plus Release 19c

This section describes a new feature introduced in SQL*Plus in 19c.

The `Easy Connect` syntax, used by applications to connect to Oracle Database, has been enhanced and is called `Easy Connect Plus`.

Deprecated Features

This section lists the deprecated features in Oracle Database release 23ai.

Deprecation of FIPS Parameters

Starting with Oracle Database 23ai, several parameters associated with `FIPS_140` are deprecated.
`FIPS_140` in `FIPS.ORA` can be used to enable `FIPS` for all features starting with Oracle Database 23ai. The following FIPS parameters are deprecated:

- SQLNET.ORA: `FIPS_140` to enable FIPS for native network encryption

- FIPS.ORA: `SSLFIPS_140` to enable FIPS for TLS

- Initialization parameter: `DBFIPS_140` to enable FIPS for TDE and `DBMS_CRYPTO`

Deprecation of Enterprise User Security (EUS)

Enterprise User Security (EUS) is deprecated with Oracle Database 23ai.
Oracle recommends that you migrate to using Centrally Managed Users (CMU). This feature enables you to directly connect with Microsoft Active Directory without an intervening directory service for enterprise user authentication and authorization to the database. If your Oracle Database is in the cloud, you can also choose to move to one of the newer integrations with a cloud identity provider.

Introduction to SQL*Plus

These instructions are to enable you to login and connect to a database after you have installed SQL*Plus. You can connect to the default database you created during installation, or to another existing Oracle database.

- SQL*Plus Resources
- SQL*Plus Overview
- Certification
- SQL*Plus Prerequisites
- Starting SQL*Plus Command-line
- About Starting SQL*Plus Instant Client
- About Connecting to a Different Database
- About Sample Schemas and SQL*Plus
- Running your first Query
- About Exiting SQL*Plus

SQL*Plus Resources

- Oracle Documentation Library at `http://www.oracle.com/technetwork/`.

SQL*Plus Overview

SQL*Plus is an interactive and batch query tool that is installed with every Oracle Database installation. It has a command-line user interface.

There is also the SQL*Plus Instant Client which is a standalone command-line interface available on platforms that support the OCI Instant Client. SQL*Plus Instant Client connects to any available Oracle database, but does not require its own Oracle database installation. See the *Oracle Call Interface Programmer's Guide* for more information on the OCI Instant Client.

SQL*Plus has its own commands and environment, and it provides access to the Oracle Database. It enables you to enter and execute SQL, PL/SQL, SQL*Plus and operating system commands to perform the following:

- Format, perform calculations on, store, and print from query results
- Examine table and object definitions
- Develop and run batch scripts
- Perform database administration

You can use SQL*Plus to generate reports interactively, to generate reports as batch processes, and to output the results to text file, to screen, or to HTML file for browsing on the Internet. You can generate reports dynamically using the HTML output facility of SQL*Plus.

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 the Oracle Database.

The SQL*Plus language is easy to use. For example, to rename a column labeled LAST_NAME with the heading "Family Name", enter the command:

```
COLUMN LAST_NAME HEADING 'Family Name'
```

Similarly, to list column definitions for the EMPLOYEES table, enter the command:

```
DESCRIBE EMPLOYEES
```

How Can I Learn SQL*Plus

There are several sources available to assist you to learn SQL*Plus:

- Part II of this Guide, Using SQL*Plus

- Help for SQL*Plus, Command-line help

- Oracle Database 12c: SQL Fundamentals

  An instructor-led course run by Oracle. This is a comprehensive hands-on course taking the student through all aspects of using SQL*Plus to access Oracle Database.

- More Oracle Database 12c Training

  To find more useful Oracle courses, go to `http://www.oracle.com/education`.

How to Use the SQL*Plus Guide

This guide provides information about SQL*Plus that applies to all operating systems. It also includes some Windows and UNIX specific information. Some aspects of SQL*Plus may differ on each operating system. Operating system specific details are covered in the Oracle Database Installation Guide provided for your system. Use these operating system specific guides with this *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. The tables are described in "About Sample Schemas and SQL*Plus".

SQL*Plus Command-line Architecture

SQL*Plus command-line uses a two-tier model comprising:

- Client (command-line user interface).

- Database (Oracle Database).

The two tiers may be on the same computer.

SQL*Plus Client

The command-line user interface is the character-based terminal implementation.

Oracle Database

Oracle Database Net components provide communication between the SQL*Plus Client and Oracle Database.

Certification

SQL*Plus is certified against the operating systems set out in the operating-system specific Oracle Database documentation.

SQL*Plus is certified against all the supported versions of Oracle Database and Oracle Server.

SQL*Plus Prerequisites

SQL*Plus is a component of Oracle Database. SQL*Plus is installed by default when you install the Oracle Database.

Some aspects of Oracle Database and SQL*Plus differ from one computer and operating system to another. These topics are discussed in the Oracle Database Installation Guide for each operating system that SQL*Plus supports.

What is necessary before you can run SQL*Plus?

- Install Oracle Database or Oracle Client. See the Oracle Database Installation Guide for your operating system available at `http://www.oracle.com/technetwork/`.

- Obtain an Oracle Database login username and password during installation or from your Database Administrator. See Login Username and Password.

- Ensure a sample database is installed and that you have a login username and password for it. See About Sample Schemas and SQL*Plus.

- Create a default database during installation or obtain the connection identifier for the Oracle Database you want to connect to from your Database Administrator. See About Connecting to a Database.

- Ensure the database you want to connect to is started. See the STARTUP command.

SQL*Plus Date Format

The default date format in SQL*Plus is determined by the database NLS_DATE_FORMAT parameter and may use a date format displaying two digit years. You can use the SQL TO_CHAR function, or the SQL*Plus COLUMN FORMAT command in your SELECT statements to control the way dates are displayed in your report.

Starting SQL*Plus Command-line

The SQL*Plus executable is usually installed in *$ORACLE_HOME*/bin, which is usually included in your operating system *PATH* environment variable. You may need to change directory to the *$ORACLE_HOME*/bin directory to start SQL*Plus.

In the following examples, you are prompted to enter the database account password.

An example using an Easy Connection identifier to connect to the HR schema in the MYDB database running on mymachine is:

```
sqlplus hr@\"//mymachine.mydomain:port/MYDB\"
```

An example using a Net Service Name is:

```
sqlplus hr@MYDB
```

Net Service Names can be stored in several places, including Oracle Names. See the Net Services Reference Guide for more information.

If you want to use Net Service Names configured in a local Oracle Net tnsnames.ora file, then set the environment variable TNS_ADMIN to the directory containing the tnsnames.ora file. For example, on UNIX, if your tnsnames.ora file is in /home/user1 and it defines the Net Service Name MYDB2:

```
TNS_ADMIN=/home/user1
export TNS_ADMIN
sqlplus hr@MYDB2
```

This example assumes the ORACLE_HOME environment variable is set, and the $ORACLE_HOME/network/admin/tnsnames.ora or ORACLE_HOME\network\admin\tnsnames.ora file defines the Net Service Name MYDB3:

```
sqlplus hr@MYDB3
```

The TWO_TASK (on UNIX) or LOCAL (on Windows) environment variable can be set to a connection identifier. This removes the need to explicitly enter the connection identifier whenever a connection is made in SQL*Plus or SQL*Plus Instant Client. This UNIX example connects to the database known as MYDB4:

```
TNS_ADMIN=/home/user1
export TNS_ADMIN
TWO_TASK=MYDB4
export TWO_TASK
sqlplus hr
```

To start SQL*Plus and connect to the default database

1. Open a UNIX or a Windows terminal and enter the SQL*Plus command:

   ```
   sqlplus
   ```

2. When prompted, enter your Oracle Database username and password. If you do not know your Oracle Database username and password, ask your Database Administrator.

3. Alternatively, enter the SQL*Plus command in the form:

   ```
   sqlplus username
   ```

You are prompted to enter your password.

4. SQL*Plus starts and connects to the default database.

Now you can start entering and executing SQL, PL/SQL and SQL*Plus statements and commands at the SQL> prompt.

Example 1   To start SQL*Plus and connect to a database other than the default

Open a UNIX or a Windows terminal and enter the SQL*Plus command:

```
sqlplus username@connect_identifier
```

You are prompted to enter your password.

Using SQL*Plus from Oracle Instant Client Package

SQL*Plus can be installed from Oracle Instant Client packages. For information about using it, see Starting SQL*Plus Command-line.

Because SQL*Plus Instant Client does not include a database, it is always 'remote' from any database server. To connect to a database you must specify the database using an Oracle Net connection identifier.

If TNS_ADMIN is not set, then an operating system dependent set of directories is examined to find tnsnames.ora. This search path includes looking in the directory specified by the ORACLE_HOME environment variable for network/admin/tnsnames.ora. This is the only reason to set the ORACLE_HOME environment variable for SQL*Plus Instant Client. If ORACLE_HOME is set when running Instant Client applications, it must be set to a directory that exists.

About Connecting to a Different Database

From an existing command-line session, enter a CONNECT command in the form:

```
SQL> connect username@connect_identifier
```

You are prompted to enter your password.

About Sample Schemas and SQL*Plus

Sample schemas are no longer included with the Oracle Database. Starting from Oracle Database 21c, you need to download sample schemas from the GitHub repository. Examples in this guide use the EMP_DETAILS_VIEW view of the Human Resources (HR) sample schema. This schema contains personnel records for a fictitious company. To view column details for the view, EMP_DETAILS_VIEW, enter

```
DESCRIBE EMP_DETAILS_VIEW
```

For more information about the sample schemas, see the *Oracle Database Sample Schemas* guide.

Unlocking the Sample Tables

The Human Resources (HR) Sample Schema is not installed as part of the default Oracle Database installation. Starting from Oracle Database 21c, you need to download the sample schemas from the GitHub repository. The HR account is locked by default.

You must unlock the HR account before you can use the HR sample schema. To unlock the HR account, log in as the SYSTEM user and enter the following command, where *your_password* is the password you want to define for the user HR:

```
ALTER USER HR IDENTIFIED BY your_password ACCOUNT UNLOCK;
```

For further information about unlocking the HR account, see the *Oracle Database Sample Schemas* guide. The HR user is primarily to enable you to access the HR sample schema and is necessary to enable you to run the examples in this guide.

Each table in the database is "owned" by a particular user. You may want 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 HR tables, see your DBA or see the *Oracle Database Sample Schemas* guide, or you can create the HR tables with the script HR_MAIN.SQL which is located in the following directory on UNIX:

```
$ORACLE_HOME/demo/schema/human_resources/hr_main.sql
```

And on the following directory on Windows:

```
ORACLE_HOME\DEMO\SCHEMA\HUMAN_RESOURCES\HR_MAIN.SQL
```

To create the HR tables from command-line SQL*Plus, do the following:

1. Ask your DBA for your Oracle Database account username and password.
2. Login to SQL*Plus.
3. On UNIX, enter the following command at the SQL*Plus prompt:

   ```
   SQL> @?/DEMO/SCHEMA/HUMAN_RESOURCES/HR_MAIN.SQL
   ```

   On Windows, enter the following command at the SQL*Plus prompt:

   ```
   SQL> @?\DEMO\SCHEMA\HUMAN_RESOURCES\HR_MAIN.SQL
   ```

To remove the sample tables, perform the same steps but substitute HR_DROP.SQL for HR_MAIN.SQL.

Running your first Query

To describe a database object, for example, column details for EMP_DETAILS_VIEW, enter a DESCRIBE command like:

```
DESCRIBE EMP_DETAILS_VIEW
```

which produces the following output:

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
JOB_ID	NOT NULL	VARCHAR2(10)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)
LOCATION_ID		NUMBER(4)
COUNTRY_ID		CHAR(2)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
JOB_TITLE	NOT NULL	VARCHAR2(35)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_NAME		VARCHAR2(40)
REGION_NAME		VARCHAR2(25)

To rename the column headings, and to select data from the HR sample schema view, EMP_DETAILS_VIEW, enter

```
COLUMN FIRST_NAME HEADING "First Name"
COLUMN LAST_NAME HEADING "Family Name"
SELECT FIRST_NAME, LAST_NAME
FROM EMP_DETAILS_VIEW
WHERE LAST_NAME LIKE 'K%';
```

which produces the following output:

First Name	Family Name
Payam	Kaufling
Steven	King
Neena	Kochhar
Alexander	Khoo
Janette	King
Sundita	Kumar

About Exiting SQL*Plus

To exit SQL*Plus command-line, enter EXIT.

Part I

Introduction to SQL*Plus

Part 1 provides the information you need to get started with SQL*Plus. It describes the command-line user interface, provides configuration information and information you need to log in and run SQL*Plus.

Part 1 contains the following chapters:

- SQL*Plus User Interface
- Configuring SQL*Plus
- Starting SQL*Plus

ORACLE®

1

SQL*Plus User Interface

This chapter describes the SQL*Plus command-line user interface. It contains the following topics:

- About The Command-line Screen
- Changing the Command-line Font and Font Size

1.1 About the Command-line Screen

The SQL*Plus command-line interface is standard on all operating systems.

When SQL*Plus starts, it displays the date and time, the SQL*Plus version and copyright information before the SQL*Plus prompt appears. The default prompt for SQL*Plus command-line is:

```
SQL>
```

1.2 Changing the Command-line Font and Font Size

In Windows, from a Command Prompt, open the Command Prompt Properties dialog to set the font and font size used in the SQL*Plus command-line interface.

To Change the Command-line Interface Font and Font Size

1. Right click in the command-line interface title bar.

2. Click Properties. The Window Preview box displays the current window's relative size on your monitor based on your font and font size selections. The Selected Font: box displays a sample of the current font.

3. Click the Font tab.

4. Select the font size to use from the Size box. Raster font sizes are shown as width by height in pixels. TrueType font sizes are shown as height in pixels.

5. Select the font to use from the Font box.

6. Select the Bold Fonts check box to use a bold version of the font.

For more information about changing Command Prompt properties, see Windows Help or click Help in the Command Prompt Properties dialog.

2

Configuring SQL*Plus

This chapter explains how to configure your SQL*Plus command-line environment. It has the following topics:

- SQL*Plus Environment Variables
- SQL*Plus Configuration

2.1 SQL*Plus Environment Variables

These environment variables specify the location or path of files used by SQL*Plus. For other environment variables that influence the behavior of SQL*Plus, see the *Oracle Database Administrator's Reference*.

Table 2-1   *Parameters or Environment Variables influencing SQL*Plus*

Parameter or Variable	Description
LD_LIBRARY_PATH	Environment variable to specify the path used to search for libraries on UNIX and Linux. The environment variable may have a different name on some operating systems, such as DYLD_LIBRARY_PATH on Apple Mac OS, LIBPATH on IBM/AIX-5L, and SHLIB_PATH on HP-UX. Not applicable to Windows operating systems.
	Example
	`$ORACLE_HOME/lib`
LOCAL	Windows environment variable to specify a connection string. Performs the same function as TWO_TASK on UNIX.
NLS_LANG	Environment variable to specify globalization behavior.
	Example
	`american_america.utf8`
ORACLE_HOME	Environment variable to specify where SQL*Plus is installed. It is also used by SQL*Plus to specify where message files are located.
	Examples:
	`d:\oracle\10g` `/u01/app/oracle/product/v10g`

Table 2-1   (Cont.) *Parameters or Environment Variables influencing SQL*Plus*

Parameter or Variable	Description
ORA_EDITION	Environment variable to specify the database edition to use. If you specify the edition with the CONNECT or SQLPLUS command option, *edition=value*, it is used instead of ORA_EDITION. If no edition is specified in either the CONNECT or SQLPLUS command option, or in ORA_EDITION, SQL*Plus connects to the default edition. When ORA_EDITION is set, a subsequent STARTUP command in the session results in an ORA-38802 error. To correct this, you must unset ORA_EDITION, then reconnect and shutdown the database, then start the database again.
ORA_NLS10	Environment variable to specify the locations of the NLS data and the user boot file in SQL*Plus 10.2. The default location is $ORACLE_HOME/nls/data. In a system with both Oracle9*i* and 10*g*, or a system under version upgrade, you should set ORA_NLS10 for Oracle 10*g* and set ORA_NLS33 for 9*i*. The default NLS location in 9*i* was $ORACLE_HOME/common/nls/admin/data.
ORACLE_PATH	Environment variable to specify the location of SQL scripts. If SQL*Plus cannot find the file in ORACLE_PATH, or if ORACLE_PATH is not set, it searches for the file in the current working directory. Not applicable to Windows
ORACLE_SID	Environment variable to specify the database instance, optional
PATH	Environment variable to specify the path to search for executables, and DLLs in Windows. Typically includes ORACLE_HOME/bin
SQLPATH	Environment variable or Windows registry entry to specify the location of SQL scripts. SQL*Plus searches for SQL scripts, including *login.sql*, in the directories specified by SQLPATH. SQLPATH is a colon-separated list of directories. There is no default value set in UNIX installations. In Windows, SQLPATH is defined in a registry entry during installation. For more information about the SQLPATH registry entry, see SQLPATH Registry Entry.

TNS_ADMIN	Environment variable to specify the location of the tnsnames.ora file. If not specified, $ORACLE_HOME/network/admin is used
	
	Example
	
	`h:\network`
	`/var/opt/oracle`

TWO_TASK	UNIX environment variable to specify a connection string. Connections that do not specify a database will connect to the database specified in TWO_TASK.
	
	Example
	
	`TWO_TASK=MYDB`
	`export TWO_TASK`
	`sqlplus hr`
	
	is the same as:
	
	`sqlplus hr@MYDB`

2.1.1 SQLPATH Registry Entry

The SQLPATH registry entry specifies the location of SQL scripts. SQL*Plus searches for SQL scripts in the current directory and then in the directories specified by the SQLPATH registry entry, and in the subdirectories of SQLPATH directories.

The HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\HOME0 registry subkey (or the HOME*n* directory for the associated ORACLE_HOME) contains the SQLPATH registry entry. SQLPATH is created with a default value of ORACLE_HOME\DBS. You can specify any directories on any drive as valid values for SQLPATH.

When setting the SQLPATH registry entry, you can concatenate directories with a semicolon (;). For example:

```
c:\oracle\ora12\database;c:\oracle\ora12\dbs
```

See the Registry Editor's help system for instructions on how to edit the SQLPATH registry entry.

2.2 SQL*Plus Configuration

You can set up your SQL*Plus environment to use the same settings with each session.

There are two operating system files to do this:

- The Site Profile file, *glogin.sql*, for site wide settings.
- Additionally, the User Profile, *login.sql*, sets user specific settings.

The exact names of these files is system dependent.

> **✎ Note:**
>
> The Site Profile and User Profile files are run after a successful Oracle Database connection from a `SQLPLUS` or `CONNECT` command, or where `/NOLOG` is specified.The Site Profile and User Profile files are not run when you switch to another PDB using `ALTER SESSION SET CONTAINER`.

Some privileged connections may generate errors if SET SERVEROUTPUT or SET APPINFO commands are put in the Site Profile or User Profile.

The following tables show the profile scripts, and some commands and settings that affect the Command-line user interface.

Table 2-2   Profile Scripts affecting SQL*Plus User Interface Settings

This script ...	is run in the Command-line...
Site Profile (glogin.sql) Can contain any content that can be included in a SQL*Plus script, such as system variable settings or other global settings the DBA wants to implement.	After successful Oracle Database connection from a SQLPLUS or CONNECT command. Where /NOLOG is specified.
User Profile (login.sql) Can contain any content that can be included in a SQL*Plus script, but the settings are only applicable to the user's sessions.	Immediately after the Site Profile.

Table 2-3   Commands in Profile scripts affecting SQL*Plus User Interface Settings

In a profile script, this command ...	affects the Command-line by ...
SET SQLPLUSCOMPAT[IBILITY] {x.y[.z]} Also see the SQL*Plus Compatibility Matrix.	Setting the SQL*Plus compatibility mode to obtain the behavior the DBA wants for this site.
SQLPLUS command COMPATIBILITY Option	As for SET SQLPLUSCOMPATIBILITY but set with the SQLPLUS command COMPATIBILITY option.
SQLPLUS command RESTRICT Option	Starting SQL*Plus with the RESTRICT option set to 3 prevents the User Profile script from being read.

2.2.1 Site Profile

A Site Profile script is created during installation. It is used by the database administrator to configure site-wide behavior for SQL*Plus Command-line connections. The Site Profile script installed during installation is an empty script.

The Site Profile script is generally named glogin.sql. SQL*Plus executes this script whenever a user starts a SQL*Plus session and successfully establishes the Oracle Database connection.

The Site Profile enables the DBA to set up site wide SQL*Plus environment defaults for all users of a particular SQL*Plus installation

Users cannot directly access the Site Profile.

2.2.1.1 Default Site Profile Script

The Site Profile script is $ORACLE_HOME/sqlplus/admin/glogin.sql in UNIX, and ORACLE_HOME\sqlplus\admin\glogin.sql in Windows. If a Site Profile already exists at this location, it is overwritten when you install SQL*Plus. If SQL*Plus is removed, the Site Profile script is deleted.

2.2.2 User Profile

For SQL*Plus command-line connections, SQL*Plus also supports a User Profile script. The User Profile is executed after the Site Profile and is intended to allow users to specifically customize their session. The User Profile script is generally named login.sql. SQL*Plus searches for the directories you specify with the ORACLE_PATH environment variable. SQL*Plus searches this colon-separated list of directories and their subdirectories in the order they are listed.

> **Note:**
>
> SQL*Plus will no longer search for login.sql in the current directory.

You can add any SQL commands, PL/SQL blocks, or SQL*Plus commands to your user profile. When you start SQL*Plus, it automatically searches for your user profile and runs the commands it contains.

2.2.2.1 Modifying Your LOGIN File

You can modify your LOGIN file just as you would any other script. The following sample User Profile script shows some modifications that you could include:

```
--   login.sql
--   SQL*Plus user login startup file.
--
--   This script is automatically run after glogin.sql
--
-- To change the SQL*Plus prompt to display the current user,
-- connection identifier and current time.
-- First set the database date format to show the time.
ALTER SESSION SET nls_date_format = 'HH:MI:SS';

-- SET the SQLPROMPT to include the _USER, _CONNECT_IDENTIFIER
-- and _DATE variables.
SET SQLPROMPT "_USER'@'_CONNECT_IDENTIFIER _DATE> "

-- To set the number of lines to display in a report page to 24.
SET PAGESIZE 24

-- To set the number of characters to display on each report line to 78.
SET LINESIZE 78

-- To set the number format used in a report to $99,999.
SET NUMFORMAT $99,999
```

> **See Also:**
>
> - SET command for more information on these and other SET command variables you may wish to set in your SQL*Plus LOGIN file.
>
> - About Using Predefined Variables for more information about predefined variables.

2.2.3 Storing and Restoring SQL*Plus System Variables

From the Command-line you can store the current SQL*Plus system variables in a script with the STORE command. If you alter any variables, this script can be run to restore the original values. This is useful if you want to reset system variables after running a report that alters them. You could also include the script in your User Profile script so that these system variables are set each time you start SQL*Plus.

To store the current setting of all system variables, enter

```
STORE SET file_name
```

Enter a file name and file extension, or enter only the file name to use the default extension .SQL. You can use the SET SUF[FIX] {SQL | text} command to change the default file extension.

2.2.3.1 Restoring the System Variables

To restore the stored system variables, enter

```
START file_name
```

If the file has the default extension (as specified by the SET SUF[FIX] {SQL | text} 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 script.

```
Created file plusenv
```

Now the value of any system variable can be changed:

```
SHOW PAGESIZE
```

```
PAGESIZE 24
```

```
SET PAGESIZE 60
SHOW PAGESIZE
```

```
PAGESIZE 60
```

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

```
START plusenv
SHOW PAGESIZE
```

```
PAGESIZE 24
```

Example 2-1   Storing and Restoring SQL*Plus System Variables

To store the current values of the SQL*Plus system variables in a new script "plusenv.sql":

```
STORE SET plusenv
```

2.2.4 About Installing Command-line Help

Command-line help is usually installed during Oracle Database installation. If not, the database administrator can create the SQL*Plus command-line help tables and populate them with SQL*Plus help data by running a supplied SQL script from SQL*Plus.

The database administrator can also remove the SQL*Plus command-line help tables by running a SQL script from SQL*Plus.

Before you can install or remove SQL*Plus help, ensure that:

- SQL*Plus is installed.
- The ORACLE_HOME environment variable is set.
- The SQL*Plus help script files exist:
  - HLPBLD.SQL - to drop and create new help tables.
  - HELPDROP.SQL - to drop existing help tables.
  - HELPUS.SQL - to populate the help tables with the help data.

2.2.4.1 Running the hlpbld.sql Script to Install Command-line Help

Run the provided SQL script, HLPBLD.SQL, to load command-line help.

1. Log in to SQL*Plus as the SYSTEM user with:

   ```
   SQLPLUS SYSTEM
   ```

   You are prompted to enter the password you have defined for the SYSTEM user.

2. In UNIX run the SQL script, HLPBLD.SQL, from SQL*Plus with:

   ```
   @$ORACLE_HOME/sqlplus/admin/help/hlpbld.sql helpus.sql
   ```

   In Windows run the SQL script, HLPBLD.SQL, from SQL*Plus with:

   ```
   @%ORACLE_HOME%\SQLPLUS\ADMIN\HELP\HLPBLD.SQL HELPUS.SQL
   ```

   The HLPBLD.SQL script creates and loads the help tables.

2.2.4.2 Running the helpdrop.sql Script to Remove Command-line Help

Run the provided SQL script, HELPDROP.SQL, to remove the command-line help.

1. Log in to SQL*Plus as the SYSTEM user with:

   ```
   SQLPLUS SYSTEM
   ```

   You are prompted to enter the password you have defined for the SYSTEM user.

2. In UNIX run the SQL script, HELPDROP.SQL, from SQL*Plus with:

   ```
   @$ORACLE_HOME/sqlplus/admin/help/helpdrop.sql
   ```

   In Windows run the SQL script, HELPDROP.SQL, from SQL*Plus with:

   ```
   @%ORACLE_HOME\SQLPLUS\ADMIN\HELP\HELPDROP.SQL
   ```

   The HELPDROP.SQL script drops the help tables, and then disconnects.

2.2.5 About Configuring Oracle Net Services

If you plan to connect to a database other than the default, whether on the same computer or another computer, you need to ensure that Oracle Net is installed, and the database listener is configured and running. Oracle Net services are used by SQL*Plus.

Oracle Net services and the database listener are installed by default during Oracle Database installation. For further information about installing and configuring Oracle Net, see the Oracle Database documentation at `http://www.oracle.com/technology/documentation`.

3

Starting SQL*Plus

This chapter describes how to start, login, and connect to a database, how to get help, and how to exit SQL*Plus.

Specific topics discussed are:

- Login Username and Password
- About Connecting to a Database
- About Starting SQL*Plus
- About Exiting SQL*Plus Command-line
- SQLPLUS Program Syntax

3.1 Login Username and Password

When you start SQL*Plus, you need a username and password to login to an Oracle Database schema. Your username and password identify you as an authorized user of the Oracle Database schema.

The database administrator (DBA) is responsible for creating your database account with the necessary privileges and giving you the username and password that enables you to access your account.

Default logins are created and you are prompted for associated passwords during Oracle Database installation. Some of the default login usernames created are:

- SYS
- SYSTEM
- HR

Logins are created and displayed in messages during Oracle Database installation.

For further information about the default logins, see Types of Oracle Database Users.

Once you have logged in, you can connect under a different username using the 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

```
CONNECT TODD
```

You are prompted to enter the password, FOX.

In the command-line interface, if you omit the username and password, SQL*Plus prompts you for them. 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, you are prompted 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 use the DISCONNECT command to disconnect from a database without leaving SQL*Plus.

3.1.1 Secure External Password Store

As a command-line alternative for large-scale deployments where applications use password credentials to connect to databases, it is possible to store such credentials in a client-side Oracle wallet. An Oracle wallet is a secure software container that is used to store authentication and signing credentials.

Storing database password credentials in a client-side Oracle wallet eliminates the need to embed usernames and passwords in application code, batch jobs, or scripts. This reduces the risk of exposing passwords in the clear in scripts and application code, and simplifies maintenance because you need not change your code each time usernames and passwords change. In addition, not having to change application code also makes it easier to enforce password management policies for these user accounts.

When you configure a client to use the external password store, applications can use the following syntax to connect to databases that use password authentication:

```
CONNECT /@database_alias
```

Note that you need not specify database login credentials in this CONNECT statement. Instead your system looks for database login credentials in the client wallet.

> ✎ **See Also:**
>
> *Oracle Database Administrator's Guide* for information about configuring your client to use secure external password store and for information about managing credentials in it.

3.1.2 Expired Password

In the command-line interface, if your password has expired, SQL*Plus prompts you to change it when you attempt to log in. You are logged in once you successfully change your password.

3.1.3 About Changing your Password

In the command-line interface, you can change your password with the PASSWORD command. See PASSWORD.

3.2 About Connecting to a Database

You must connect to an Oracle Database (instance) before you can query or modify data in that database. You can connect to the default database and to other databases accessible through your network. To connect to another database over a network, both databases must have Oracle Net configured, and have compatible network drivers.

You must enter either a connection identifier or a net service name to connect to a database other than the default.

The connection identifier or net service name is entered:

- as an argument to the SQLPLUS Program Syntax when starting a command-line session.
- as an argument to the CONNECT command from a current session. For detailed usage, see *Accessing a Container in a CDB* in the *Oracle Database Administrator's Guide*.

3.2.1 Net Service Name

Your DBA is responsible for creating the databases you use and defining net service names for them in the *tnsnames.ora* file.

A net service name definition in the *tnsnames.ora* file has the syntax:

```
net_service_name=
(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=port) )
(CONNECT_DATA=
(SERVICE_NAME=service_name) ) )
```

To use a net service name (alias), it must have an entry in the *tnsnames.ora* file on the machine running SQL*Plus. An entry in *tnsnames.ora* is not required if you use a connection identifier.

Example 3-1   The tnsnames.ora entry for the sales database

```
SALES1 =
(DESCRIPTION =
(ADDRESS=(PROTOCOL=tcp)(HOST=sales-server)(PORT=1521) )
(CONNECT_DATA=
(SERVICE_NAME=sales.us.acme.com) ) )
```

Example 3-2   Start a command-line session to the sales database using the net service name

```
SQLPLUS hr@SALES1
```

See Configuration Parameters and Configuration and Administration of Oracle Net Services for more information about database connections and net service name definitions.

3.2.2 Full Connection Identifier

Depending on your configuration, use the full connection identifier syntax like:

```
(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=port) )
(CONNECT_DATA=
(SERVICE_NAME=service_name) ) )
```

The SERVICE_NAME is the global database name entered during database creation. It combines a database name with a domain name. For example, the SERVICE_NAME sales.us.acme.com has a database name of sales and a domain of us.acme.com.

An INSTANCE_NAME is the name you give to the database instance during creation. It defaults to the SID you entered during database creation.

An Oracle System Identifier (`SID`) identifies a specific Oracle release 8.0 database instance.

You can optionally use an `INSTANCE_NAME` in place of the `SERVICE_NAME` phrase.

Use a `SID` in place of the `SERVICE_NAME` when connecting to an Oracle release 8.0 or earlier database.

Example 3-3   Full connection identifier for SALES1

```
SQLPLUS hr@\"(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcp)(HOST=sales-server)(PORT=1521) )
(CONNECT_DATA=
(SERVICE_NAME=sales.us.acme.com) ) )\"
```

3.2.3 Easy Connection Identifier

The easy or abbreviated connection identifier has the syntax:

```
[//]host[:port][/service_name]
```

Example 3-4   Start a command-line session to the sales database using the easy connection identifier

```
sqlplus hr@\"sales-server:1521/sales.us.acme.com\"
```

Example 3-5   CONNECT to the sales database using the easy connection identifier

When the password is omitted, the connect string needs to be quoted.

```
connect hr@"sales-server:1521/sales.us.acme.com"
```

The easy connection identifier can be used wherever you can use a full connection identifier, or a net service name. The easy syntax is less complex, and no *tnsnames.ora* entry is required.

> **Note:**
>
> Starting with Oracle Database release 19c, the easy connect syntax has improved functionality and is called Easy Connect Plus. For more information, see About Easy Connect Plus.

3.2.4 Connectionless Session with /NOLOG

In the command-line interface, it is possible to start SQL*Plus without connecting to a database. This is useful for performing some database administration tasks, writing transportable scripts, or to use SQL*Plus editing commands to write or edit scripts.

You use the /NOLOG argument to the SQLPLUS command to start a connectionless command-line session. After SQL*Plus has started you can connect to a database with the CONNECT command.

Example 3-6   Start a connectionless SQL*Plus session with /NOLOG

```
SQLPLUS /NOLOG
```

3.3 About Starting SQL*Plus

If you are connecting to a remote Oracle database, make sure your Oracle Net software is installed and working properly. For more information, see Testing and Troubleshooting Oracle Net Services.

When you start a SQL*Plus command-line session, and after a CONNECT command in that session, the site profile, *glogin.sql*, and the user profile file, *login.sql*, are processed:

- After SQL*Plus starts and connects, and prior to displaying the first prompt.
- After SQL*Plus starts and connects, and prior to running a script specified on the command line.
- Prior to the first prompt when /NOLOG is specified on the command line and no connection is made.

The site profile file, *glogin.sql* is processed first, then the user profile file, *login.sql*.

3.3.1 About Starting Command-line SQL*Plus

To begin using SQL*Plus, you must first understand how to start and stop SQL*Plus.

1. Make sure that SQL*Plus has been installed on your computer.
2. Log on to the operating system (if required).
3. Enter the command, SQLPLUS, and press Return.

> ✏ **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 current date, and copyright information, and prompts you for your username (the text displayed on your system may differ slightly):

```
SQL*Plus: Release 21.0.0.0.0 Production on Fri Nov 13 22:12:47 2020
Version 21.1.0.0.0

Copyright (c) 1982, 2020, Oracle. All rights reserved.
```

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 Database to which you connected and the versions of available tools such as PL/SQL, and the local time of the last time you logged on.

```
SQL*Plus: Release 21.0.0.0.0 Production on Fri Nov 13 22:12:47 2020
Version 21.1.0.0.0

Copyright (c) 1982, 2020, Oracle. All rights reserved.

Last Successful login time: Fri Nov 06 2020 23:35:38 -07:00

Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - 64bit
Production
```

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

```
SQL>
```

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

If SQL*Plus does not start, you should see a message to help you correct the problem.

Example 3-7   Starting SQL*Plus

This example shows you how to start SQL*Plus:

3.3.2 About Getting Command-line Help

To access command-line help for SQL*Plus commands, type HELP or ? followed by the command name at the SQL command prompt or in the *i*SQL*Plus Workspace Input area. See the HELP command for more information. For example:

```
HELP ACCEPT
```

To display a list of SQL*Plus commands, type HELP followed by either TOPICS or INDEX. HELP TOPICS displays a single column list of SQL*Plus commands. HELP INDEX displays a four column list of SQL*Plus commands which fits in a standard screen. For example:

```
HELP INDEX
```

3.4 About Exiting SQL*Plus Command-line

If you cannot log in to SQL*Plus because your username or password is invalid or for some other reason, SQL*Plus returns an error status equivalent to an EXIT FAILURE command. See the EXIT command for further information.

When you are done working with SQL*Plus and wish to return to the operating system, enter EXIT or QUIT at the SQL*Plus prompt, or enter the end of file character, Ctrl+D on UNIX or Ctrl+Z on Windows.

SQL*Plus displays the version of Oracle Database from which you disconnected and the versions of tools available through SQL*Plus before you return to the operating system prompt.

3.5 SQL*Plus Program Syntax

You use the SQLPLUS command at the operating system prompt to start command-line SQL*Plus:

```
SQLPLUS [ [Options] [Logon|/NOLOG] [Start] ]
```

where: Options has the following syntax:

```
-H[ELP]|-V[ERSION]
|[[-C[OMPATIBILITY] {x.y[.z]] [-F[ast]] [-M[ARKUP] markup_option] [-L[OGON]]
 [-NOLOGINTIME] [-P <connect identifier>] [-R[ESTRICT] {1|2|3}] [-S[ILENT]]]
```

and *markup_option* consists of:

- *csv_option*

- *html_option*

csv_option has the following syntax:

```
CSV {ON|OFF} [DELIMI[TER] character] [QUOTE {ON|OFF}]
```

html_option has the following syntax:

```
HTML [ON|OFF] [HEAD text] [BODY text] [TABLE text] [ENTMAP {ON|OFF}] [SPOOL {ON|OFF}]
[PRE[FORMAT] {ON|OFF}]
```

where Logon has the following syntax:

```
{username[/password][@connect_identifier]| / }
[AS {SYSASM|SYSBACKUP|SYSDBA|SYSDG|SYSOPER|SYSRAC|SYSKM}][edition=value]
```

where Start has the following syntax:

```
@{url|file_name[.ext]} [arg ...]
```

> **⚠ WARNING:**
>
> Including your password in plain text is a security risk. You can avoid this risk by omitting the password, and entering it only when the system prompts for it.

You have the option of entering logon. If you do not specify logon but do specify start, SQL*Plus assumes that the first line of the script contains a valid logon. If neither start nor logon are specified, SQL*Plus prompts for logon information.

3.5.1 Options

The following sections contain descriptions of SQLPLUS command options:

3.5.1.1 HELP Option

```
-H[ELP]
```

Displays the usage and syntax for the SQLPLUS command, and then returns control to the operating system.

3.5.1.2 VERSION Option

```
-V[ERSION]
```

Displays the current version and level number for SQL*Plus, and then returns control to the operating system.

3.5.1.3 COMPATIBILITY Option

```
-C[OMPATIBILITY] {x.y[.z]
```

Sets the value of the SQLPLUSCOMPATIBILITY system variable to the SQL*Plus release specified by $x.y[.z]$. Where x is the version number, y is the release number, and z is the update number. For example, 9.0.1 or 10.2. For more information, see the SET SQLPLUSCOMPAT[IBILITY] {x.y[.z]}system variable.

3.5.1.4 LOGON Option

```
-L[OGON]
```

Specifies not to reprompt for username or password if the initial connection does not succeed. This can be useful in operating system scripts that must either succeed or fail and you don't want to be reprompted for connection details if the database server is not running.

3.5.1.5 FAST Option

```
-F[ast]
```

The FAST option improves general performance. This command line option changes the values of the following default settings:

- ARRAYSIZE = 100
- LOBPREFETCH = 16384
- PAGESIZE = 50000
- ROWPREFETCH = 2
- STATEMENTCACHE = 20

3.5.1.6 MARKUP Options

```
-M[ARKUP]
```

You can use the MARKUP options to generate output in HTML or CSV (Character Separated Values) format, through a query or script.

MARKUP currently supports HTML 4.0 transitional, and the CSV format.

Use SQLPLUS -MARKUP to produce output in HTML or CSV format.

> **✎ Note:**
>
> Depending on your operating system, the complete *markup_option* clause for the
> SQL*PLUS command may need to be contained in quotes.

For HTML output, use SQLPLUS -MARKUP HTML ON or SQLPLUS -MARKUP HTML ON
SPOOL ON to produce standalone web pages. SQL*Plus will generate complete HTML
pages automatically encapsulated with <HTML> and <BODY> tags. The HTML tags in a
spool file are closed when SPOOL OFF is executed or SQL*Plus exits.

The -SILENT and -RESTRICT command-line options may be useful when used in
conjunction with -MARKUP.

You can use MARKUP HTML ON to produce HTML output in either the <PRE> tag or in an
HTML table. Output to a table uses standard HTML <TABLE>, <TR> and <TD> tags to
automatically encode the rows and columns resulting from a query. Output to an HTML table
is the default behavior when the HTML option is set ON. You can generate output using
HTML <PRE> tags by setting PREFORMAT ON.

For CSV output, use SQLPLUS -MARKUP CSV ON to produce output in CSV format. You
can specify the delimiter character by using the DELIMITER option. You can also output text
without quotes by using QUOTE OFF.

Use the SHOW MARKUP command to view the status of MARKUP options.

The SQLPLUS -MARKUP command has the same functionality as the SET MARKUP
command. These options are described in this section. For other information on the SET
MARKUP command, see the SET command.

```
CSV {ON|OFF}
```

CSV is a mandatory MARKUP argument which specifies that the type of output to be
generated is CSV. The optional CSV arguments, ON and OFF, specify whether or not to
generate CSV output. The default is OFF. You can turn CSV output ON and OFF as required
during a session.

```
HTML {ON|OFF}
```

HTML is a mandatory MARKUP argument which specifies that the type of output to be
generated is HTML. The optional HTML arguments, ON and OFF, specify whether or not to
generate HTML output. The default is OFF.

MARKUP HTML ON generates HTML output using the specified MARKUP options.

You can turn HTML output ON and OFF as required during a session.

```
HEAD text
```

The HEAD *text* option enables you to specify content for the <HEAD> tag. By default, *text*
includes a default in-line cascading style sheet and title.

If *text* includes spaces, it must be enclosed in quotes. SQL*Plus does not test this free text
entry for HTML validity. You must ensure that the text you enter is valid for the HTML
<HEAD> tag. This gives you the flexibility to customize output for your browser or special
needs.

```
BODY text
```

The BODY *text* option enables you to specify attributes for the <BODY> tag. By default, there are no attributes. If *text* includes spaces, it must be enclosed in quotes. SQL*Plus does not test this free text entry for HTML validity. You must ensure that the text you enter is valid for the HTML <BODY> tag. This gives you the flexibility to customize output for your browser or special needs.

```
TABLE text
```

The TABLE *text* option enables you to enter attributes for the <TABLE> tag. You can use TABLE *text* to set HTML <TABLE> tag attributes such as BORDER, CELLPADDING, CELLSPACING and WIDTH. By default, the <TABLE> WIDTH attribute is set to 90% and the BORDER attribute is set to 1.

If *text* includes spaces, it must be enclosed in quotes. SQL*Plus does not test this free text entry for HTML validity. You must ensure that the text you enter is valid for the HTML <TABLE> tag. This gives you the flexibility to customize output for your browser or special needs.

```
ENTMAP {ON|OFF}
```

ENTMAP ON or OFF specifies whether or not SQL*Plus replaces special characters <, >, " and & with the HTML entities <, >, " and & respectively. ENTMAP is set ON by default.

You can turn ENTMAP ON and OFF as required during a session. For example, with ENTMAP OFF, SQL*Plus screen output is:

```
SQL>PROMPT A > B
A > B
```

With ENTMAP ON, SQL*Plus screen output is:

```
SQL&gt; PROMPT A > B
A &gt; B
```

As entities in the <HEAD> and <BODY> tags are not mapped, you must ensure that valid entities are used in the MARKUP HEAD and BODY options.

If entities are not mapped, web browsers may treat data as invalid HTML and all subsequent output may display incorrectly. ENTMAP OFF enables users to write their own HTML tags to customize output.

> **Note:**
>
> ENTMAP only takes effect when the HTML option is set ON. For more information about using entities in your output, see the COLUMN command.

```
SPOOL {ON|OFF}
```

SPOOL ON or OFF specifies whether or not SQL*Plus writes the HTML opening tags, <HTML> and <BODY>, and the closing tags, </BODY> and </HTML>, to the start and end of each file created by the SQL*Plus SPOOL filename command. The default is OFF.

You can turn SPOOL ON and OFF as required during a session.

> **✎ Note:**
>
> It is important to distinguish between the SET MARKUP HTML SPOOL option, and the SQLPLUS SPOOL *filename* command.
>
> The SET MARKUP HTML SPOOL ON option enables the writing of the <HTML> tag to the spool file. The spool file is not created, and the header and footer tags enabled by the SET MARKUP HTML SPOOL ON option are not written to the spool file until you issue the SQLPLUS SPOOL *filename* command. See the SPOOL command for more information.

SQL*Plus writes several HTML tags to the spool file when you issue the SPOOL filename command.

When you issue any of the SQL*Plus commands: EXIT, SPOOL OFF or SPOOL filename, SQL*Plus appends the following end tags and closes the file:

```
</BODY>
</HTML>
```

You can specify <HEAD> tag contents and <BODY> attributes using the HEAD and BODY options

```
PRE[FORMAT] {ON|OFF}
```

PREFORMAT ON or OFF specifies whether or not SQL*Plus writes output to the <PRE> tag or to an HTML table. The default is OFF, so output is written to a HTML table by default. You can turn PREFORMAT ON and OFF as required during a session.

> **✎ Note:**
>
> To produce report output using the HTML <PRE> tag, you must set PREFORMAT ON. For example:
>
> ```
> SQLPLUS -M "HTML ON PREFORMAT ON"
> ```
>
> or
>
> ```
> SET MARKUP HTML ON PREFORMAT ON
> ```

3.5.1.7 MARKUP Usage Notes

MARKUP HTML ON

When MARKUP HTML ON PREFORMAT OFF is used, commands originally intended to format paper reports have different meaning for reports intended for web tables:

- PAGESIZE is the number of rows in an HTML table, not the number of lines. Each row may contain multiple lines. The TTITLE, BTITLE and column headings are repeated every PAGESIZE rows.

- LINESIZE may have an effect on data if wrapping is on, or for very long data. Depending on data size, output may be generated on separate lines, which a browser may interpret as a space character.

- TTITLE and BTITLE content is output to three line positions: left, center and right, and the maximum line width is preset to 90% of the browser window. These elements may not align with the main output as expected due to the way they are handled for web output. Entity mapping in TTITLE and BTITLE is the same as the general ENTMAP setting specified in the MARKUP command.

- If you use a title in your output, then SQL*Plus starts a new HTML table for output rows that appear after the title. Your browser may format column widths of each table differently, depending on the width of data in each column.

- SET COLSEP, RECSEP and UNDERLINE only produce output in HTML reports when PREFORMAT is ON.

MARKUP CSV ON

When MARKUP CSV ON is used, output from a query will be displayed in CSV format.

You can enable CSV markup while logging into a user session, by using the -M[ARKUP] CSV ON option at the SQL*Plus command line. For more information, see SQL*Plus Program Syntax. While logged in to a user session, you can enable CSV markup by using the SET MARKUP CSV ON command.

You can specify the delimiter character by using the DELIMITER option. You can also output text without quotes by using QUOTE OFF.

3.5.1.8 No Login Time Option

```
-nologintime
```

The last login time for non-SYS users is displayed when you log on. This feature is on by default. The last login time is displayed in local time format. You can use the –nologintime option to disable this security feature. After you login, the last login information is displayed

```
SQL*Plus: Release 21.0.0.0.0 Production on Fri Nov 13 22:12:47 2020
Version 21.1.0.0.0

Copyright (c) 1982, 2020, Oracle. All rights reserved.

Last Successful login time: Fri Nov 06 2020 23:35:38 -07:00

Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.1.0.0.0
With the Partitioning, Oracle Label Security, Data Mining and Real Application
```

Last login time does not show when making a connection with the CONNECT command.

3.5.1.9 PING Option

```
-P connect_identifer
```

Pings the network listener associated with the provided connect identifier and displays a success or error message along with one of the following exit codes:

- 0: when the ping is successful

- 1: when the ping fails

See the PING command for more information.

3.5.1.10 RESTRICT Option

```
-R[ESTRICT] {1|2|3}
```

Enables you to disable certain commands that interact with the operating system. This is similar to disabling the same commands in the Product User Profile (PUP) table. However, commands disabled with the -RESTRICT option are disabled even if there is no connection to a server, and remain disabled until SQL*Plus terminates.

If no -RESTRICT option is active, than all commands can be used, unless disabled in the PUP table.

If -RESTRICT 3 is used, then LOGIN.SQL is not read. GLOGIN.SQL is read but restricted commands used will fail.

Table 3-1   Commands Disabled by Restriction Level

Command	Level 1	Level 2	Level 3
EDIT	disabled	disabled	disabled
GET			disabled
HOST	disabled	disabled	disabled
SAVE		disabled	disabled
SPOOL		disabled	disabled
START, @, @@			disabled
STORE		disabled	disabled

3.5.1.11 SILENT Option

```
-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. If you omit username or password, SQL*Plus prompts for them, but the prompts are not visible! Use SILENT to invoke SQL*Plus within another program so that the use of SQL*Plus is invisible to the user.

SILENT is a useful mode for creating reports for the web using the SQLPLUS -MARKUP command inside a CGI script or operating system script. The SQL*Plus banner and prompts are suppressed and do not appear in reports created using the SILENT option.

3.5.2 Logon

```
username[/password]
```

Represent the username and password with which you wish to start SQL*Plus and connect to Oracle Database.

> **⚠ WARNING:**
>
> Including your password in plain text is a security risk. You can avoid this risk by omitting the password, and entering it only when the system prompts for it.

If you omit username and password, SQL*Plus prompts you for them. If you omit only password, SQL*Plus prompts for it. In silent mode, username and password prompts are not visible! Your username appears when you type it, but not your password.

`@`*`connect_identifier`*

Consists of an Oracle Net connect identifier. The exact syntax depends upon the Oracle Net configuration. For more information, refer to the Oracle Net manual or contact your DBA.

`edition=`*`value`*

The value for the Oracle Edition. An edition enables two or more versions of an object in a database. It provides a staging area where changed objects can be loaded into the database, compiled, and executed during uptime. This is particularly useful to reduce downtime associated with patching an application. *edition=value* overrides any edition value specified in the ORA_EDITION environment variable. For more detailed information, see Using Edition-Based Redefinition.

`/`

Represents a default logon using operating system authentication. You cannot enter a connect identifier if you use a default logon. In a default logon, SQL*Plus typically attempts to log you in using the username OPS$*name*, where name is your operating system username. Note that the prefix "OPS$" can be set to any other string of text. For example, you may wish to change the settings in your INIT.ORA parameters file to LOGON*name* or USERID*name*. See Using Operating System Authentication for information about operating system authentication.

`AS {SYSASM |SYSBACKUP |SYSDBA |SYSDG |SYSOPER |SYSRAC |SYSKM}`

The AS clause enables privileged connections by users who have been granted SYSASM, SYSBACKUP, SYSDBA, SYSDG, SYSOPER, SYSRAC or SYSKM system privileges.

`/NOLOG`

Establishes no initial connection to Oracle Database. 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 script prompt for the username, password, or database specification. The first line of this script is not assumed to contain a logon.

3.5.3 Start

`@{`*`url`*`|`*`file_name`*`[.`*`ext`*`]} [`*`arg ...`*`]`

Specifies the name of a script and arguments to run. The script can be called from the local file system or from a web server.

SQL*Plus passes the arguments to the script as if executing the file using the SQL*Plus START command. If no file suffix (file extension) is specified, the suffix defined by the SET SUFFIX command is used. The default suffix is .sql.

See the START command for more information.

Part II
Using SQL*Plus

Part II helps you learn how to use SQL*Plus, how to tune SQL*Plus for better performance, how to restrict access to tables and commands and provides overviews of database administration tools and globalization support.

Part II contains the following chapters:

- SQL*Plus Basics
- Using Scripts in SQL*Plus
- Using Substitution Variables
- Formatting SQL*Plus Reports
- Generating HTML Reports from SQL*Plus
- Tuning SQL*Plus
- SQL*Plus Security
- Database Administration with SQL*Plus
- SQL*Plus Globalization Support

4

SQL*Plus Basics

This chapter helps you learn the basics of using SQL*Plus. It has the following topics:

- About Entering and Executing Commands
- About Listing a Table Definition
- About Listing PL/SQL Definitions
- Running SQL Commands
- About Running PL/SQL Blocks
- Running SQL*Plus Commands
- System Variables that Affect How Commands Run
- About Stopping a Command while it is Running
- About Running Operating System Commands
- About Pausing the Display
- About Saving Changes to the Database Automatically

4.1 About Entering and Executing Commands

Unless stated otherwise, descriptions of commands are applicable to all user interfaces.

In the command-line, type commands at the SQL*Plus prompt and press Return to execute them.

Usually, you separate the words in a command with a space or a tab. You can use additional spaces or tabs between words to make your commands more readable.

Case sensitivity is operating system specific. 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:

- 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.

4.1.1 The SQL Buffer

The SQL buffer stores the most recently entered SQL command or PL/SQL block (but not SQL*Plus commands). The command or block remains in the buffer until replaced by the next SQL command or PL/SQL block. You can view the buffer contents with the LIST command.

You can execute the command or block in the SQL buffer using the RUN or /(slash) commands. RUN displays the command or block in the buffer before executing it. / (slash) executes the command or block in the buffer without displaying it first. For information about editing a command or block stored in the buffer see About Editing Scripts in SQL*Plus Command-Line.

SQL*Plus does not store the following in the SQL buffer:

- SQL*Plus commands
- Trailing white space
- Semicolon or slash characters you type to execute a command

4.1.2 About Executing Commands

In command-line SQL*Plus, you type a command and direct SQL*Plus to execute it by pressing the Return key. SQL*Plus processes the command and re-displays the command prompt when ready for another command.

4.2 About Listing a Table Definition

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

```
Name                                     Null?    Type
---------------------------------------- -------- ----------------
EMPLOYEE_ID                              NOT NULL NUMBER(6)
JOB_ID                                   NOT NULL VARCHAR2(10)
MANAGER_ID                                        NUMBER(6)
DEPARTMENT_ID                                     NUMBER(4)
LOCATION_ID                                       NUMBER(4)
COUNTRY_ID                                        CHAR(2)
FIRST_NAME                                        VARCHAR2(20)
LAST_NAME                                NOT NULL VARCHAR2(25)
SALARY                                            NUMBER(8,2)
COMMISSION_PCT                                    NUMBER(2,2)
DEPARTMENT_NAME                          NOT NULL VARCHAR2(30)
JOB_TITLE                                NOT NULL VARCHAR2(35)
CITY                                     NOT NULL VARCHAR2(30)
STATE_PROVINCE                                    VARCHAR2(25)
COUNTRY_NAME                                      VARCHAR2(40)
REGION_NAME                                       VARCHAR2(25)
```

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

Example 4-1   Using the DESCRIBE Command

To list the column definitions of the columns in the sample view EMP_DETAILS_VIEW, enter

```
DESCRIBE EMP_DETAILS_VIEW
```

4.3 About Listing PL/SQL Definitions

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

Example 4-2   Using the DESCRIBE Command

To create and list the definition of a function called AFUNC, enter

```
create or replace function afunc (f1 varchar2, f2 number) return number as
begin
   if (length(f1) > f2) then
    return 1;
   else
    return 0;
   end if;
end;
/

FUNCTION created.


DESCRIBE afunc


FUNCTION afunc RETURNS NUMBER
Argument Name    Type      In/Out   Default?
---------------- -------- -------- ---------
F1               VARCHAR2 IN
F2               NUMBER   IN
```

4.4 Running SQL Commands

The SQL command language enables you to manipulate data in the database. See your *Oracle Database SQL Language Reference* for information on individual SQL commands.

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

    ```
    SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
    ```

    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.

ORACLE®

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

```
FROM EMP_DETAILS_VIEW WHERE SALARY > 12000;
```

The semicolon (;) means that this is the end of the command. Press Return or click Execute. SQL*Plus processes the command and displays the results:

```
EMPLOYEE_ID LAST_NAME                 JOB_ID         SALARY
----------- ------------------------- ---------- --------------
        100 King                      AD_PRES        $24,000
        101 Kochhar                   AD_VP          $17,000
        102 De Haan                   AD_VP          $17,000
        145 Russell                   SA_MAN         $14,000
        146 Partners                  SA_MAN         $13,500
        201 Hartstein                 MK_MAN         $13,000

6 rows selected.
```

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

The headings may be repeated in your output, depending on the setting of a system variable called PAGESIZE. Sometimes, the result from a query will not fit the available page width. You can use the system variable, LINESIZE, to set the width of the output in characters. See Setting Page Dimensions. Typically, LINESIZE is set to 80 in command-line. Whether you see the message stating the number of records retrieved depends on the setting of the system variable, FEEDBACK. See System Variables that Affect How Commands Run for more information.

Example 4-3   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 EMP_DETAILS_VIEW view.

4.4.1 About 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.

4.4.1.1 About 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. Thus, you can enter the query you entered in Example 4-3 on three lines:

```
SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
```

In this guide, you will find most SQL commands divided into clauses, one clause on each line. In Example 4-3, for instance, the SELECT and FROM clauses were placed

on separate lines. Many people find this clearly visible structure helpful, but you may choose whatever line division makes commands most readable to you.

4.4.1.2 About 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 4-3, and press Return or click Execute. SQL*Plus processes the command and also stores the command in the SQL buffer. See The SQL Buffer for details. If you mistakenly press Return before typing the semicolon, SQL*Plus prompts you with a line number for the next line of your command. Type the semicolon and press Return again or click Execute to run the command.

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 or click Execute. SQL*Plus executes the command and stores it in the buffer.

A blank line in a SQL statement or script 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.

> **Note:**
>
> You can change the way blank lines appear and behave in SQL statements using the SET SQLBLANKLINES command. For more information about changing blank line behavior, see the SET command.

To execute commands this way, 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 for details. If you subsequently enter another SQL command, SQL*Plus overwrites the previous command in the buffer.

4.5 About Running PL/SQL Blocks

You can also use PL/SQL subprograms (called blocks) to manipulate data in the database. See your *Oracle Database PL/SQL Language Reference* for information on individual PL/SQL statements.

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. You can also terminate and execute a PL/SQL subprogram by entering a slash (/) by itself on a new line.

You enter the mode for entering PL/SQL statements when:

- You type DECLARE or BEGIN. 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 PROCEDURE) that creates a stored procedure. After you enter PL/SQL mode in this way, type the stored procedure you want to create.

SQL*Plus stores the subprograms you enter in the SQL buffer. Execute the current subprogram with a RUN or slash (/) command. A semicolon (;) is treated as part of the PL/SQL subprogram and will not execute the command.

SQL*Plus sends the complete PL/SQL subprogram to Oracle Database for processing (as it does SQL commands). See your *Oracle Database PL/SQL Language Reference* for more information.

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

```
DECLARE
    x   NUMBER := 100;
    BEGIN
        FOR i IN 1..10 LOOP
            IF MOD (i, 2) = 0 THEN     --i is even
              INSERT INTO temp VALUES (i, x, 'i is even');
            ELSE
              INSERT INTO temp VALUES (i, x, 'i is odd');
            END IF;
            x := x + 100;
        END LOOP;
    END;
    .
/
```

4.5.1 About Creating Stored Procedures

Stored procedures are PL/SQL functions, packages, or procedures. To create stored procedures, you use the following SQL CREATE commands:

- CREATE FUNCTION
- CREATE LIBRARY
- 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. For more information, see About Running PL/SQL Blocks. 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:

```
SHOW ERRORS PROCEDURE ASSIGNVL
```

See SHOW for more information.

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

```
EXECUTE EMPLOYEE_MANAGEMENT.NEW_EMP('BLAKE')
```

See EXECUTE for more information.

4.6 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.

To speed up command entry, you can abbreviate many SQL*Plus commands. For information on and abbreviations of all SQL*Plus commands, see SQL*Plus Command Reference.

1. Enter this SQL*Plus command:

```
COLUMN SALARY FORMAT $99,999 HEADING 'MONTHLY 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 following query and press Return to run it:

```
SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
FROM EMP_DETAILS_VIEW WHERE SALARY > 12000;


EMPLOYEE_ID LAST_NAME                 JOB_ID     MONTHLY SALARY
----------- ------------------------- ---------- --------------
        100 King                      AD_PRES           $24,000
        101 Kochhar                   AD_VP             $17,000
        102 De Haan                   AD_VP             $17,000
        145 Russell                   SA_MAN            $14,000
        146 Partners                  SA_MAN            $13,500
        201 Hartstein                 MK_MAN            $13,000

6 rows selected.
```

Example 4-4   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 SALARY of the sample view, EMP_DETAILS_VIEW.

The COLUMN command formatted the column SALARY with a dollar sign ($) and a comma (,) and gave it a new heading.

4.6.1 About Understanding SQL*Plus Command Syntax

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

You do not need to end a SQL*Plus command with a semicolon. When you finish entering the command, you can just press Return or click Execute. There is no need to end a SQL*Plus command with a semicolon.

4.6.1.1 About 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:

```
COLUMN SALARY FORMAT $99,999 -
HEADING 'MONTHLY SALARY'
```

Since SQL*Plus identifies the hyphen as a continuation character, entering a hyphen within a SQL statement is ignored by SQL*Plus. SQL*Plus does not identify the statement as a SQL statement until after the input processing has joined the lines together and removed the hyphen. For example, entering the following:

```
SELECT 200 -
100 FROM DUAL;
```

returns the error:

```
SELECT 200 100 FROM DUAL
           *
ERROR at line 1:
ORA-00923: FROM keyword not found where expected
```

To ensure that the statement is interpreted correctly, reposition the hyphen from the end of the first line to the beginning of the second line.

4.7 System Variables that Affect How Commands Run

The SQL*Plus SET command controls many variables—called SET variables or system variables—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.

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.)

See the SET command for more information on system variables and their default settings. See SQL*Plus Configuration and SQLPLUS Program Syntax for details on the SQL*Plus LOGIN file.

To list the current setting of a system variable, enter SHOW followed by the variable name. See the SHOW command for information on other items you can list with SHOW.

4.8 About 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, the system's interrupt character, which is usually CTRL+C. SQL*Plus stops the display.

> **Note:**
>
> Pressing Cancel does 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 Formatting SQL*Plus Reports.) You can stop the printing of a file through your operating system. For more information, see your operating system's installation and user's guide.

4.9 About Running Operating System Commands

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

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

```
HOST DIRECTORY *.SQL
```

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

> **Note:**
>
> Operating system commands entered from a SQL*Plus session using the HOST command do not affect the current SQL*Plus session. For example, setting an operating system environment variable does not affect the current SQL*Plus session, but may affect SQL*Plus sessions started subsequently.
>
> You can suppress access to the HOST command. For more information about suppressing the HOST command see SQL*Plus Security.

4.10 About Pausing the Display

You can use the PAUSE system variable to stop and examine the contents of the screen after each page during the display of a long report, or during the display of a table definition with many columns.

You can use SET PAUSE to pause output after displaying each screen of a query or report. See SET PAU[SE] {ON | OFF | text} for more information.

4.11 About Saving Changes to the Database Automatically

You can specify changes you wish to make to the information stored in the database using the SQL Database Manipulation Language (DML) commands UPDATE, INSERT, and DELETE—which can be used independently or within a PL/SQL block. These changes are not made permanent until you enter a SQL COMMIT command or a SQL Database Control Language (DCL) or Database Definition Language (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 AUTOCOMMIT system variable. Regardless of the AUTOCOMMIT setting, changes are committed when you exit SQL*Plus successfully.

> **See Also:**
>
> SET EXITC[OMMIT] {ON | OFF}

```
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, for example, 10, enter

```
SET AUTOCOMMIT 10
```

SQL*Plus counts SQL DML commands as they are executed and commits the changes after each 10th SQL DML command.

> **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:

```
SET AUTOCOMMIT OFF
```

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

```
SHOW AUTOCOMMIT
```

```
AUTOCOMMIT OFF
```

See SET AUTO[COMMIT]{ON | OFF | IMM[EDIATE] | n} for more information.

Example 4-5   Turning Autocommit On

To turn the autocommit feature on, enter

```
SET AUTOCOMMIT ON
```

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

```
SET AUTOCOMMIT IMMEDIATE
```

Until you change the setting of AUTOCOMMIT, SQL*Plus automatically commits changes from each SQL DML command that specifies changes to the database. After each autocommit, SQL*Plus displays the following message:

5

Using Scripts in SQL*Plus

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

- About Editing Scripts
- About Editing Scripts in SQL*Plus Command-Line
- About Placing Comments in Scripts
- Running Scripts
- Nesting Scripts
- About Exiting from a Script with a Return Code

Read this chapter while sitting at your computer and try out the examples shown. Before beginning, make sure you have access to the sample schema described in SQL*Plus Overview.

5.1 About Editing Scripts

In SQL*Plus command-line, the use of an external editor in combination with the @, @@ or START commands is an effective method of creating and executing generic scripts. You can write scripts which contain SQL*Plus, SQL and PL/SQL commands, which you can retrieve and edit. This is especially useful for storing complex commands or frequently used reports.

5.1.1 Writing Scripts with a System Editor

Your operating system may have one or more text editors that you can use to write scripts. You can run your operating system's default text editor without leaving the SQL*Plus command-line by entering the EDIT command.

You can use the SQL*Plus DEFINE command to define the variable, _EDITOR, to hold the name of your preferred text editor. For example, to define the editor used by EDIT to be vi, enter the following command:

```
DEFINE _EDITOR = vi
```

You can include an editor definition in your user or site profile so that it is always enabled when you start SQL*Plus. See SQL*Plus Configuration, and the DEFINE and EDIT commands for more information.

To create a script with a text editor, enter EDIT followed by the name of the file to edit or create, for example:

```
EDIT SALES
```

EDIT adds the filename extension .SQL to the name unless you specify the file extension. When you save the script with the text editor, it is saved back into the same file. EDIT lets you create or modify scripts.

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

Example 5-1   Using a System Editor to Write a SQL Script

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 your system editor, invoke your editor and create a file to hold your script:

```
EDIT SALES
```

Enter each of the following lines in your editor. Do not forget to include the semicolon at the end of the SQL statement:

```
COLUMN LAST_NAME HEADING 'LAST NAME'
COLUMN SALARY HEADING 'MONTHLY SALARY' FORMAT $99,999
COLUMN COMMISSION_PCT HEADING 'COMMISSION %' FORMAT 90.90
SELECT LAST_NAME, SALARY, COMMISSION_PCT
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN';
```

The format model for the column COMMISSION_PCT tells SQL*Plus to display an initial zero for decimal values, and a zero instead of a blank when the value of COMMISSION_PCT is zero for a given row. Format models and the COLUMN command are described in more detail in the COLUMN command and in Format Models.

Now use your editor's save command to store your query in a file called SALES.SQL.

5.2 About Editing Scripts in SQL*Plus Command-Line

You can use a number of SQL*Plus commands to edit the SQL command or PL/SQL block currently stored in the buffer.

Table 5-1 lists the SQL*Plus commands that allow you to examine or change the command in the buffer without re-entering the command.

Table 5-1   SQL*Plus Editing Commands

Command	Abbreviation	Purpose
APPEND *text*	A *text*	adds *text* at the end of the current line
CHANGE/old/new	C/*old*/*new*	changes *old* to *new* in the current line
CHANGE/*text*	C/*text*	deletes *text* from the current line
CLEAR BUFFER	CL BUFF	deletes all lines
DEL	(none)	deletes the current line

Table 5-1   (Cont.) SQL*Plus Editing Commands

Command	Abbreviation	Purpose
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	; or 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*

These are useful if you want to correct or modify a command you have entered.

5.2.1 Listing the Buffer Contents

The SQL buffer contains the last SQL or PL/SQL command. 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.

```
SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
  2  FROM EMP_DETAILS_VIEW
  3* WHERE SALARY>12000
```

Notice that the semicolon you entered at the end of the SELECT command is not listed. This semicolon is necessary to indicate the end of the command when you enter it, but it is not part of the SQL command and SQL*Plus does not store it in the SQL buffer.

Example 5-2   Listing the Buffer Contents

Suppose you want to list the current command. Use the LIST command as shown. (If you have exited SQL*Plus or entered another SQL command or PL/SQL block since following the steps in Example 4-3, perform the steps in that example again before continuing.)

```
LIST
```

5.2.2 Editing the Current Line

The SQL*Plus CHANGE command enables 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. (Note, that using the slash (/) command to run the command in the buffer does not affect the current line.)

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

```
SELECT EMPLOYEE_ID, LAST_NAME, JO_ID, SALARY
                                   *
ERROR at line 1:
ORA-00904: 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 mis-typed column JOB_ID.

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. You do not need to use the CHANGE command to re-enter an entire line.

```
1* SELECT EMPLOYEE_ID, FIRST_NAME, JOB_ID, SALARY
```

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

```
RUN
```

SQL*Plus correctly displays the query and its result:

```
 1  SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
 2  FROM EMP_DETAILS_VIEW
 3* WHERE JOB_ID='SA_MAN'

EMPLOYEE_ID LAST_NAME                JOB_ID     MONTHLY SALARY
----------- ------------------------ ---------- --------------
        145 Russell                  SA_MAN             $14,000
        146 Partners                 SA_MAN             $13,500
        147 Errazuriz                SA_MAN             $12,000
        148 Cambrault                SA_MAN             $11,000
        149 Zlotkey                  SA_MAN             $10,500
```

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

See CHANGE for information about the significance of case in a CHANGE command and on using wildcards to specify blocks of text in a CHANGE command.

Example 5-3   Making an Error in Command Entry

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

```
SELECT EMPLOYEE_ID, LAST_NAME, JO_ID, SALARY
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN';
```

You see this message on your screen:

Example 5-4   Correcting the Error

To change JO_ID to JOB_ID, change the line with the CHANGE command:

```
CHANGE /JO_ID/JOB_ID
```

The corrected line appears on your screen:

5.2.3 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 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 5-5   Appending Text to a Line

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

```
LIST 4
```

```
4* ORDER BY SALARY
```

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

```
APPEND  DESC


4* ORDER BY SALARY DESC
```

Type RUN to verify the query:

```
  1  SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
  2  FROM EMP_DETAILS_VIEW
  3  WHERE JOB_ID='SA_MAN'
  4* ORDER BY SALARY DESC

EMPLOYEE_ID LAST_NAME                 JOB_ID     MONTHLY SALARY
----------- ------------------------- ---------- --------------
        145 Russell                   SA_MAN            $14,000
        146 Partners                  SA_MAN            $13,500
        147 Errazuriz                 SA_MAN            $12,000
        148 Cambrault                 SA_MAN            $11,000
        149 Zlotkey                   SA_MAN            $10,500
```

5.2.4 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 all lines are renumbered starting at 1.

```
0 SELECT EMPLOYEE_ID


4
```

Enter the new line. Then press Return.

```
4 ORDER BY SALARY
```

SQL*Plus prompts you again for a new line:

```
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.

```
  1  SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY
  2  FROM EMP_DETAILS_VIEW
  3  WHERE JOB_ID='SA_MAN'
  4* ORDER BY SALARY

EMPLOYEE_ID LAST_NAME                 JOB_ID     MONTHLY SALARY
----------- ------------------------- ---------- --------------
        149 Zlotkey                   SA_MAN            $10,500
        148 Cambrault                 SA_MAN            $11,000
```

ORACLE®

```
           147 Errazuriz                 SA_MAN              $12,000
           146 Partners                  SA_MAN              $13,500
           145 Russell                   SA_MAN              $14,000
```

Example 5-6   Adding a Line

Suppose you want to add a fourth line to the SQL command you modified in Example 5-4. Since line 3 is already the current line, enter INPUT and press Return.

```
INPUT
```

SQL*Plus prompts you for the new line:

5.2.5 Deleting Lines

Use the DEL command to delete lines in the buffer. Enter DEL specifying the line numbers you want to delete.

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

```
DEL * LAST
```

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

See DEL for more information.

5.3 About Placing Comments in Scripts

You can enter comments in a script in three ways:

*   using the SQL*Plus REMARK command for single line comments.

*   using the SQL comment delimiters /*... */ for single or multi line comments.

*   using ANSI/ISO (American National Standards Institute/International Standards Organization) comments - - for single line comments.

    Comments entered at the command-line are not stored in the SQL buffer.

5.3.1 Using the REMARK Command

Use the REMARK command on a line by itself in a script, 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 Commission Report;
REMARK to be run monthly.;
COLUMN LAST_NAME HEADING 'LAST_NAME';
COLUMN SALARY HEADING 'MONTHLY SALARY' FORMAT $99,999;
COLUMN COMMISSION_PCT HEADING 'COMMISSION %' FORMAT 90.90;
REMARK Includes only salesmen;
SELECT LAST_NAME, SALARY, COMMISSION_PCT
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN';
```

5.3.2 Using /*...*/

Enter the SQL comment delimiters, /*...*/, on separate lines in your script, on the same line as a SQL command, or on a line in a PL/SQL block.

You must enter a space after the slash-asterisk(/*) beginning a comment.

The comments can span multiple lines, but cannot be nested within one another:

```
/* Commission Report
 to be run monthly. */
COLUMN LAST_NAME HEADING 'LAST_NAME';
COLUMN SALARY HEADING 'MONTHLY SALARY' FORMAT $99,999;
COLUMN COMMISSION_PCT HEADING 'COMMISSION %' FORMAT 90.90;
REMARK Includes only salesmen;
SELECT LAST_NAME, SALARY, COMMISSION_PCT
FROM EMP_DETAILS_VIEW
/* Include only salesmen.*/
WHERE JOB_ID='SA_MAN';
```

5.3.3 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
```

This comment is invalid:

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

If you enter the following SQL*Plus command, SQL*Plus interprets it as a comment and does not execute the command:

```
-- SET LONG 777
```

5.3.4 Notes on Placing Comments

SQL*Plus does not have a SQL or PL/SQL command parser. It scans the first few keywords of each new statement to determine the command type, SQL, PL/SQL or SQL*Plus. Comments in some locations can prevent SQL*Plus from correctly identifying the command type, giving unexpected results. The following usage notes may help you to use SQL*Plus comments more effectively:

1. Do not put comments within the first few keywords of a statement. For example:

```
CREATE OR REPLACE
  2  /* HELLO */
  3  PROCEDURE HELLO AS
  4  BEGIN
  5  DBMS_OUTPUT.PUT_LINE('HELLO');
  6  END;
  7  /

Warning: Procedure created with compilation errors.
```

The location of the comment prevents SQL*Plus from recognizing the command as a command. SQL*Plus submits the PL/SQL block to the server when it sees the slash "/" at the beginning of the comment, which it interprets as the "/" statement terminator. Move the comment to avoid this error. For example:

```
 CREATE OR REPLACE PROCEDURE
  2  /* HELLO */
  3  HELLO AS
  4  BEGIN
  5  DBMS_OUTPUT.PUT_LINE('HELLO');
  6  END;
  7  /

Procedure created.
```

2. Do not put comments after statement terminators (period, semicolon or slash). For example, if you enter:

```
SELECT 'Y' FROM DUAL; -- TESTING
```

You get the following error:

```
SELECT 'Y' FROM DUAL; -- TESTING
                     *
ERROR at line 1:
ORA-00911: invalid character
```

SQL*Plus expects no text after a statement terminator and is unable to process the command.

3. Do not put statement termination characters at the end of a comment line or after comments in a SQL statement or a PL/SQL block. For example, if you enter:

```
SELECT *
-- COMMENT;
```

You get the following error:

```
-- COMMENT
         *
ERROR at line 2:
ORA-00923: FROM keyword not found where expected
```

The semicolon is interpreted as a statement terminator and SQL*Plus submits the partially formed SQL command to the server for processing, resulting in an error.

4. Do not use ampersand characters '&' in comments in a SQL statement or PL/SQL block. For example, if you enter a script such as:

```
SELECT REGION_NAME, CITY
/* THIS & THAT */
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
```

SQL*Plus interprets text after the ampersand character "&" as a substitution variable and prompts for the value of the variable, &that:

```
Enter value for that:
old   2: /* THIS & THAT */
new   2: /* THIS  */

REGION_NAME              CITY
------------------------ -------------------------------
Americas                 Seattle
Americas                 Seattle
Americas                 Seattle
Europe                   Oxford
Europe                   Oxford
Americas                 Toronto
6 rows selected.
```

You can SET DEFINE OFF to prevent scanning for the substitution character.

For more information on substitution and termination characters, see DEFINE, SQLTERMINATOR and SQLBLANKLINES in the SET command.

5.4 Running Scripts

The START command retrieves a script and runs the commands it contains. Use START to run a script containing SQL commands, PL/SQL blocks, and SQL*Plus commands. You can have many commands in the file. Follow the START command with the name of the file:

```
START file_name
```

SQL*Plus assumes the file has a .SQL extension by default.

> **Note:**
>
> Starting from Oracle Database release 19c, version 19.3, executing a script that contains a $ (dollar) symbol results in an error on Windows because the $ symbol denotes an environment variable in Linux and Unix.
>
> For example:
>
> ```
> SQL>@C:\User\my$script.sql
> ```

```
LAST NAME                MONTHLY SALARY COMMISSION %
------------------------ -------------- ------------
Russell                         $14,000         0.40
```

```
Partners                            $13,500          0.30
Errazuriz                           $12,000          0.30
Cambrault                           $11,000          0.30
Zlotkey                             $10,500          0.20
```

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

```
@SALES
```

The @ and @@ commands list and run the commands in the specified script in the same manner as START. SET ECHO affects the @ and @@ commands in the same way as it affects the START command.

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

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

Example 5-7   Running a Script

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

```
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:

5.4.1 Running a Script as You Start SQL*Plus

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

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

```
SQLPLUS HR @SALES
```

  SQL*Plus starts, prompts for your password and runs the script.

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

5.5 Nesting Scripts

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

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

> **Note:**
>
> The @@ command may be useful in this example. See the @@ (double at sign) command for more information.

5.6 About Exiting from a Script with a Return Code

You can include an EXIT command in a script to return a value when the script finishes. See the EXIT command for more information.

You can include a WHENEVER SQLERROR command in a script to automatically exit SQL*Plus with a return code should your script generate a SQL error. Similarly, you can include a WHENEVER OSERROR command to automatically exit should an operating system error occur. See the WHENEVER SQLERROR command, and the WHENEVER OSERROR command for more information.

6

Using Substitution Variables

This chapter explains how SQL*Plus substitution variables work and where they can be used. It shows the relationship between the three types of variables (substitution, bind, and system) used in SQL*Plus.

This topics covered are:

- Defining Substitution Variables
- About Using Predefined Variables
- Referencing Substitution Variables
- System Variables Influencing Substitution Variables
- Passing Parameters through the START Command
- About Communicating with the User
- About Using Bind Variables
- Using REFCURSOR Bind Variables
- Fetching Iterative Results from a SELECT inside a PL/SQL Block

6.1 Defining Substitution Variables

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

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

To list all substitution variable definitions, enter DEFINE by itself. Note that any substitution 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 substitution variable of datatype NUMBER implicitly through the ACCEPT command. You will learn more about the ACCEPT command.

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

Example 6-1   Defining a Substitution Variable

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

```
DEFINE L_NAME = SMITH
```

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

```
DEFINE L_NAME
```

6.2 About Using Predefined Variables

There are nine variables containing SQL*Plus information that are defined during SQL*Plus installation. These variables can be redefined, referenced or removed the same as any other variable. They are always available from session to session unless you explicitly remove or redefine them.

> ✎ **See Also:**
>
> Predefined Variables for a list of the predefined variables and examples of their use.

6.3 Referencing Substitution Variables

Suppose you want to write a query like the one in SALES to list the employees with various jobs, not just those whose job is SA_MAN. You could do that by editing a different 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 text, SA_MAN, 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 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_ID and the variable MYTABLE has the value EMP_DETAILS_VIEW, SQL*Plus executes the commands

```
SELECT &SORTCOL, SALARY
FROM &MYTABLE
WHERE SALARY>12000;
```

as if they were

```
SELECT JOB_ID, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
```

6.3.1 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. 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 or standard input.

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.

```
Created file STATS
```

Now run the script STATS:

```
@STATS
```

And respond to the prompts for values as shown:

```
Enter value for group_col: JOB_ID
old   1: SELECT    &GROUP_COL,
new   1: SELECT    JOB_ID,
Enter value for number_col: SALARY
old   2:           MAX(&NUMBER_COL) MAXIMUM
new   2:           MAX(SALARY) MAXIMUM
Enter value for table: EMP_DETAILS_VIEW
old   3: FROM      &TABLE
new   3: FROM      EMP_DETAILS_VIEW
Enter value for group_col: JOB_ID
old   4: GROUP BY &GROUP_COL
new   4: GROUP BY JOB_ID
```

SQL*Plus displays the following output:

```
JOB_ID        MAXIMUM
---------- ----------
AC_ACCOUNT       8300
AC_MGR          12000
AD_ASST          4400
AD_PRES         24000
AD_VP           17000
FI_ACCOUNT       9000
FI_MGR          12000
HR_REP           6500
IT_PROG          9000
MK_MAN          13000
MK_REP           6000


JOB_ID        MAXIMUM
---------- ----------
PR_REP          10000
PU_CLERK         3100
PU_MAN          11000
SA_MAN          14000
SA_REP          11500
SH_CLERK         4200
ST_CLERK         3600
ST_MAN           8200

19 rows selected.
```

A more practical use of substitution variables is to prompt for a value before referencing the variable:

```
SQL> accept myv char prompt 'Enter a last name: '
SQL> select employee_id from employees where last_name = '&myv';
```

If these two commands are stored in a SQL*Plus script, a different last name can be entered each time the script is run.

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

```
SELECT SALARY FROM EMP_DETAILS_VIEW WHERE EMPLOYEE_ID='&X.5';
Enter value for X:   20
```

is interpreted as

```
SELECT SALARY FROM EMP_DETAILS_VIEW WHERE EMPLOYEE_ID='205';
```

If you want to append a period immediately after a substitution variable name, then use two periods together. For example, if "myfile" is defined as "reports" then the command:

```
SQL> spool &myfile..log
```

is the same as:

```
SQL> spool reports.log
```

Text in ANSI "/* */" or "--" comments that looks like a substitution variable may be treated as one. For example:

```
SQL> select department_id, location_id /* get dept & loc */ from
departments;
Enter value for loc: _
```

Here the text "& loc" in the comment is interpreted as a variable reference. SQL*Plus prompts you for a value for the variable "loc".

Example 6-2   Using Substitution Variables

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

```
SELECT &GROUP_COL, MAX(&NUMBER_COL) MAXIMUM
FROM &TABLE
GROUP BY &GROUP_COL
.
SAVE STATS
```

6.3.2 Difference Between "&" and "&&" Prefixes

Both single ampersand (&) and double ampersand (&&) can prefix a substitution variable name in a statement. SQL*Plus pre-processes the statement and substitutes

ORACLE®

the variable's value. The statement is then executed. If the variable was not previously defined then SQL*Plus prompts you for a value before doing the substitution.

If a single ampersand prefix is used with an undefined variable, the value you enter at the prompt is not stored. Immediately after the value is substituted in the statement the variable is discarded and remains undefined. If the variable is referenced twice, even in the same statement, then you are prompted twice. Different values can be entered at each prompt:

```
SQL> prompt Querying table &mytable
Enter value for mytable: employees
Querying table employees
SQL> select employee_id from &mytable where last_name = 'Jones';
Enter value for mytable: employees


EMPLOYEE_ID
-----------
        195
```

If a double ampersand reference causes SQL*Plus to prompt you for a value, then SQL*Plus defines the variable as that value (that is, the value is stored until you exit). Any subsequent reference to the variable (even in the same command) using either "&" or "&&" substitutes the newly defined value. SQL*Plus will not prompt you again:

```
SQL> prompt Querying table &&mytable
Enter value for mytable: employees
Querying table employees
SQL> select employee_id from &mytable where last_name = 'Jones';


EMPLOYEE_ID
-----------
        195
```

6.3.3 Storing a Query Column Value in a Substitution Variable

Data stored in the database can be put into substitution variables:

```
SQL> column last_name new_value mynv
SQL> select last_name from employees where employee_id = 100;
```

The NEW_VALUE option in the COLUMN command implicitly creates a substitution variable called *mynv*. The variable is not physically created until a query references the column LAST_NAME. When the query finishes, the variable *mynv* holds the last retrieved value from the column LAST_NAME:

```
SQL> define mynv
DEFINE mynv       = "King" (CHAR)
```

6.3.4 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. The

buffer editing commands, APPEND, CHANGE, and INPUT, treat text beginning with "&" or "&&" literally, like any other text string.

6.3.5 How Substitution Variables are Handled in SQL*Plus

Substitution variable references are pre-processed and substituted before the command is parsed and executed. For each statement, SQL*Plus will do the following:

```
1. Loop for each "&" and "&&" variable reference:
    If the variable already has a value defined (i.e. stored)
        Replace the variable reference with the value
    else
        Prompt for a value
        Replace the variable reference with the value
        If the variable is prefixed with "&&" then
            define (i.e. store) the variable for future use

2. Execute the statement
```

Step 1 happens inside the SQL*Plus client tool. SQL*Plus then sends the final statement to the database engine where step 2 occurs.

It is not possible to repeatedly prompt in a PL/SQL loop. This example prompts once and the entered value is substituted in the script text. The resulting script is then sent to the database engine for execution. The same entered value is stored five times in the table:

```
begin
  for i in 1 .. 5 loop
    insert into mytable values (&myv);
  end loop;
end;
/
```

Substitution variables are not recursively expanded. If the value of a referenced variable contains an ampersand, then the ampersand is used literally and is not treated as a second variable prefix:

```
SQL> set escape \
SQL> define myv = \&mytext
SQL> prompt &myv
&mytext
```

You cannot use a substitution variable as the first token of a command. Each command name must be hard-coded text else an error is displayed. For example:

```
SQL> &myv * from dual;
SP2-0734: unknown command beginning "&myv * fro..." - rest of line
ignored.
```

6.3.6 Substitution Variable Commands

Substitution variables can be used to replace options and values in almost all SQL*Plus commands. Several of the commands have special significance for substitution variables.

Command	Description
ACCEPT	Reads a line of input and stores it in a given substitution variable.
COLUMN	Specifies display attributes for a given column.
DEFINE	Specifies a user or predefined variable and assigns a CHAR value to it, or lists the value and variable type of a single variable or all variables.
EDIT	Invokes an operating system text editor on the contents of the specified file or on the contents of the buffer.
EXIT	Commits or rolls back all pending changes, logs out of Oracle Database, terminates SQL*Plus and returns control to the operating system.
HOST	Executes an operating system command without leaving SQL*Plus.
TTITLE, BTITLE, REPHEADER, REPFOOTER	TTITLE places and formats a specified title at the top of each report page. BTITLE places and formats a specified title at the bottom of each report page. REPHEADER places and formats a specified report header at the top of each report. REPFOOTER places and formats a specified report footer at the bottom of each report.
UNDEFINE	Deletes one or more substitution variables that you defined either explicitly (with the DEFINE command) or implicitly (with an argument to the START command).
WHENEVER	WHENEVER OSERROR performs the specified action (exits SQL*Plus by default) if an operating system error occurs (such as a file writing error). WHENEVER SQLERROR performs the specified action (exits SQL*Plus by default) if a SQL command or PL/SQL block generates an error.

See SQL*Plus Command Summary for more information about these substitution variable commands.

6.3.6.1 Using "&" Prefixes With Title Variables

The title commands (TTITLE, BTITLE, REPHEADER and REPFOOTER) substitute variables differently to most other commands. (The exceptions are the EXIT and SET SQLPROMPT commands, which are similar to the title commands).

The guidelines for variables in titles are:

- If you want the same value for a variable to be printed on every page then use an "&" prefix and put the variable inside a quoted string:

```
accept mycustomer char prompt 'Enter your company name: '
ttitle left 'Report generated for company &mycustomer'
select last_name, job_id from employees order by job_id;
```

- If you want each title to have data from the query that is unique to each report page then do not use an "&" prefix for the variable and do not put the variable inside quotes.

```
column job_id new_value ji_nv noprint
break on job_id skip page
ttitle left 'Employees in job: ' ji_nv
select last_name, job_id from employees order by job_id;
```

SQL*Plus substitution variables are expanded before each command is executed. After this happens in a title command, the resulting string is stored as the title text. What makes variables in titles special is that they need to be re-substituted for each page of query results. This is so the current COLUMN NEW_VALUE and OLD_VALUE substitution variable values are displayed on each page, customizing each title for the results displayed on its page. If "&" is used inadvertently or incorrectly to prefix title variables, it is possible to get double substitution. This is dependent on the variable's value and is easily overlooked when you write scripts.

Any non-quoted, non-keyword in a title is checked when the page is printed to see if it is a variable. If it is, its value is printed. If not, then the word is printed verbatim. This means that if you use "&myvar" in a title command, and the text substituted for it can itself be interpreted as another variable name then you get double variable substitution. For example, the script:

```
define myvar = scottsvar
ttitle left &myvar
define scottsvar = Hello
select * from dual;
```

causes the text "left scottsvar" to be stored as the title. When the title is printed on each page of the query this string is re-evaluated. The word "scottsvar" in the title is itself treated as a variable reference and substituted. The query output is:

```
Hello
D
-
deX
```

Using "&" in titles most commonly causes a problem with the numeric variable names of the SQL*Plus script parameters. If the value of an arbitrary "&"-prefixed title variable is the same as a script parameter variable name, then double substitution will occur.

To display an "&" in a title, prefix it with the SET ESCAPE character. The ampersand (&) is stored as the title text and is not substituted when page titles are printed.

6.3.6.2 Variables and Text Spacing in Titles

Unquoted whitespace in titles is removed. Use whitespace instead of the SET CONCAT character to separate variables from text that should appear immediately adjacent. Use whitespace inside quotes to display a space. For example, the script:

```
define myvar = 'ABC'
ttitle left myvar myvar Text ' Other words'
select ...;
```

gives a title of:

```
ABCABCText Other words
```

6.3.7 Substitution Variable Namespace, Types, Formats and Limits

Substitution Variable Namespace

In a SQL*Plus session there is just one global name space for substitution variables. If you reconnect using CONNECT, or run subscripts using "@", all variables ever defined are available for use and may be overridden or undefined.

When a child script finishes, all substitution variables it defined or changed are visible to the calling script. This is particularly noticeable when a subscript executed with "@" or START is given script parameters. The parameters "&1" etc. get redefined and the parent script sees the new values.

To minimize problems, and for general readability, use symbolic variable names for command parameters. All other references should use the new variable name instead of "&1". For example:

```
define myuser = '&1'
@myscript.sql King
select first_name from employees where last_name = '&myuser';
```

The call to *myscript.sql* changes the value of "&1" to "King". By saving the original value of "&1" in "myuser" and using "&myuser" instead of "&1" in the SELECT, the query executes correctly.

Substitution Variable Types

The substitution variable types stored by SQL*Plus are:

- CHAR
- NUMBER
- BINARY_FLOAT
- BINARY_DOUBLE

The CHAR type is a generic text format similar to the database table VARCHAR2 column type. All variables created from the following are of type CHAR:

- with DEFINE
- from prompts for "&" variables
- from script parameters

This ensures that values entered are substituted verbatim with no conversion loss.

Variables created by COLUMN NEW_VALUE or OLD_VALUE for the columns in Oracle number format will have the type NUMBER. These substitution variables are stored in Oracle's internal number representation as they are in the database. This allows display formats to be altered without any internal value loss. Substitution variables of BINARY_FLOAT and BINARY_DOUBLE types are similarly created for Oracle BINARY_FLOAT and BINARY_DOUBLE columns. These variables are stored in native machine representation. The CHAR type is used for NEW_VALUE and OLD_VALUE variables with all other column types.

There is no explicit DATE type. The DATE keyword in the ACCEPT command is used solely to allow correct format validation against a date format. Substitution variables created by ACCEPT ... DATE, or by COLUMN NEW_VALUE on a date column, are stored as type CHAR. For example:

```
SQL> accept mydvar date format 'DD-MON-YYYY'
prompt 'Enter a date: '
Enter a date: 03-APR-2003
SQL> define mydvar
DEFINE MYDVAR               = "03-APR-2003" (CHAR)
```

If a variable already exists and is redefined, its old type is discarded and the new type used.

The type of a substitution variable is generally transparent. Substitution variables are weakly typed. For example, a COLUMN NEW_VALUE variable takes on the particular type of the named column in each new query. It may also change type during a query. For example, the type of a substitution variable used on a NUMBER column changes from NUMBER to CHAR when a NULL value is fetched. It changes back to NUMBER when the next numeric value is fetched.

No type comparison semantics are defined for any type since there is no direct comparison of variables. All variables are textually substituted before any SQL or PL/SQL statement that could do a comparison is executed.

Substitution Variable Formats

When a variable is substituted, or its value is shown by a DEFINE command, it is formatted as text before the command referencing the variable is finally executed.

CHAR variables are substituted verbatim.

NUMBER variables are formatted according to SET NUMWIDTH (by default) or SET NUMFORMAT (if you have explicitly set one):

The display format of a number can be changed even after the variable is created. To show this, first create a NUMBER variable. You cannot use DEFINE to do this because it makes the type of all new variables CHAR. Instead use a COLUMN NEW_VALUE command which inherits the NUMBER type from a NUMBER column:

```
SQL> column c2 new_val m
SQL> select 1.1 c2 from dual C2;
----------
1.1
```

```
SQL> define m
DEFINE M              =         1.1 (NUMBER)
```

Changing the format affects the display of the number but not the stored value:

```
SQL> set numformat 99.990
SQL> define m
DEFINE M              =   1.100 (NUMBER)
```

Substitution Variable Limits

The maximum number of substitution variables allowed is 2048. SQL*Plus gives an error an attempt is made to create more. The limit includes the predefined variables, however these can be undefined if necessary. Leaving a large number of unnecessarily defined variables can reduce the performance of SQL*Plus because variable lookups are slower.

A character substitution variable can be up to 240 bytes long.

A numeric substitution variable holds the full range of Oracle numbers.

When a command line undergoes variable substitution, the resulting line length can be no more than:

- 3000 bytes if it is a line of SQL (like SELECT or INSERT) or PL/SQL text (like BEGIN or CREATE PROCEDURE)

- 2499 bytes if it a line of a SQL*Plus command (like TTITLE or COLUMN)

Otherwise an error is displayed.

These limits may be lower in old versions of SQL*Plus.

6.3.8 Assigning Substitution Variables to Bind Variables

You can assign a substitution variable to a bind variable:

```
SQL> define mysubv = 123
SQL> variable mybndv number
SQL> execute :mybndv := &mysubv;
```

SQL*Plus executes the PL/SQL assignment statement after it substitutes the value of "mysubv". If "mysubv" was not already defined, you would be prompted for a value.

The bind variable can be used in subsequent SQL or PL/SQL commands.

6.3.9 Assigning Bind Variables to Substitution Variables

Sometimes it is useful to make the value of a bind variable available to SQL*Plus commands like TTITLE or SPOOL. For example, you might want to call a PL/SQL function that returns a string and use the value for a SQL*Plus spool file name. The SPOOL command does not understand bind variable syntax so the bind variable value needs to be assigned to a substitution variable first.

This is done using COLUMN NEW_VALUE and SELECT commands. For example, declare a bind variable in SQL*Plus and instantiate it in a PL/SQL block. Its value can be returned from a PL/SQL function, or like here, set by a direct assignment:

```
SQL> variable mybv varchar2(14)
SQL> begin
  2    /* ... */
  3    :mybv := 'report.log';
  4  end;
  5  /
```

Pass the bind variable's value to a new substitution variable "nv" by using a query:

```
SQL> column mybvcol new_value nv noprint
SQL> select :mybv mybvcol from dual;
```

Now you can use the substitution variable in a SPOOL command:

```
SQL> spool &nv
```

The SPOOL command executes as if you had typed

```
SQL> spool report.log
```

6.3.10 Substitution Variable Examples

The following examples demonstrate how to use substitution variables.

- Setting a Substitution Variable's Value
- Using a Substitution Variable
- Finding All Defined Substitution Variables
- Inserting Data Containing "&" Without Being Prompted
- Putting the Current Date in a Spool File Name
- Appending Alphanumeric Characters Immediately After a Substitution Variable
- Putting a Period After a Substitution Variable
- Using a Fixed Value Variable in a TTITLE, BTITLE, REPHEADER or REPFOOTER
- Using a Changing Value Variable in a TTITLE, BTITLE, REPHEADER or REPFOOTER
- Using the Value of a Bind Variable in a SQL*Plus Command Like SPOOL
- Passing Parameters to SQL*Plus Substitution Variables
- Passing Operating System Variables to SQL*Plus
- Passing a Value to a PL/SQL Procedure From the Command Line
- Allowing Script Parameters to be Optional and Have a Default Value
- Using a Variable for the SQL*Plus Return Status

- [Putting the Username and Database in the Prompt](#)

6.3.10.1 Setting a Substitution Variable's Value

A substitution variable can be set in several ways. The common ways are as follows:

- The DEFINE command sets an explicit value:

```
define myv = 'King'
```

- The ACCEPT command:

```
accept myv char prompt 'Enter a last name: '
```

prompts you for a value and creates a character variable "myv" set to the text you enter.

- Using "&&" before an undefined variable prompts you for a value and uses that value in the statement:

```
select first_name from employees where last_name = '&&myuser';
```

If the substitution variable "myuser" is not already defined, then this statement creates "myuser" and sets it to the value you enter.

- Using COLUMN NEW_VALUE to set a substitution variable to a value stored in the database:

```
column last_name new_value mynv    select last_name from employees where employee_id = 100;
```

This creates a substitution variable "mynv" set to the value in the "last_name" column.

6.3.10.2 Using a Substitution Variable

Once a substitution variable has a value, it can be referenced by prefixing the variable name with an ampersand (&).

If the variable "myv" is already defined, it can be used as:

```
select employee_id from employees where last_name = '&myv';
```

6.3.10.3 Finding All Defined Substitution Variables

The DEFINE command with no parameters shows all defined substitution variables, their values, and their types. For example:

```
define
```

might give:

```
DEFINE MYV              = "King" (CHAR)
...
```

6.3.10.4 Inserting Data Containing "&" Without Being Prompted

There are two ways to make an "&" be treated as text and not cause a prompt. The first turns all variable substitution off:

```
set define off
create table mytable (c1 varchar2(20));
insert into mytable (c1) values ('thick & thin');
```

The INSERT statement stores the text "thick & thin" in the table.

The second method is useful for ignoring individual occurrences of "&" while allowing others to prefix substitution variables:

```
set escape \
create table mytable (c1 varchar2(20));
insert into mytable (c1) values ('thick \& thin');
insert into mytable (c1) values ('&mysubvar');
```

The first INSERT statement in this method stores the text "thick & thin" in the table. The second INSERT causes SQL*Plus to prompt you for a value, which is then stored.

6.3.10.5 Putting the Current Date in a Spool File Name

Using SYSDATE you can query the current date and put it in a substitution variable. The substitution variable can then be used in a SPOOL command:

```
column dcol new_value mydate noprint
select to_char(sysdate,'YYYYMMDD') dcol from dual;
spool &mydate.report.txt

-- my report goes here
select last_name from employees;

spool off
```

In this example, the first query puts the date in the substitution variable "mydate". There is no visible output from this query because of the NOPRINT option in the COLUMN command. In the SPOOL command, the first period (.) indicates the end of the variable name and is not included in the resulting string. If "mydate" contained "20030120" from the first query, then the spool file name would be "20030120report.txt".

You can use this technique to build up any string for the file name.

The period is the default value of SET CONCAT. If you have assigned another character, use it instead of a period to end the substitution variable name.

6.3.10.6 Appending Alphanumeric Characters Immediately After a Substitution Variable

If you wish to append alphanumeric characters immediately after a substitution variable, use the value of SET CONCAT to separate the variable name from the following text. The default value of SET CONCAT is a single period (.). For example:

```
define mycity = Melbourne
spool &mycity.Australia.txt
```

creates a file with the name "MelbourneAustralia.txt".

6.3.10.7 Putting a Period After a Substitution Variable

If SET CONCAT is a period (.) and you want to append a period immediately after a substitution variable, use two periods together. For example:

```
define mycity = Melbourne
spool &mycity..log
```

is the same as:

```
spool Melbourne.log
```

6.3.10.8 Using a Fixed Value Variable in a TTITLE, BTITLE, REPHEADER or REPFOOTER

This example makes every page of a report have exactly the same heading. It can be used for TTITLE, BTITLE, REPHEADER or REPFOOTER commands. In a TTITLE command, prefix the variable name "dept" with "&" and place it inside a quoted string:

```
define dept = '60'
ttitle left 'Salaries for department &dept'
select last_name, salary from employees where department_id = &dept;
```

6.3.10.9 Using a Changing Value Variable in a TTITLE, BTITLE, REPHEADER or REPFOOTER

This example uses a different title on every page of a report. Each title contains a value derived from query results shown on that particular page. In a TTITLE command, do not put an "&" before the variable name "dv". Put the variable name outside a quoted string:

```
column department_id new_value dv noprint
ttitle left 'Members of department ' dv
break on department_id skip page
```

```
select department_id, last_name from employees order by department_id,
last_name;
```

In a BTITLE or REPFOOTER command, use a COLUMN OLD_VALUE variable instead of a COLUMN NEW_VALUE variable.

6.3.10.10 Using the Value of a Bind Variable in a SQL*Plus Command Like SPOOL

If you want to use the value of a bind variable in a SQL*Plus command, it must first be copied to a substitution variable.

SQL*Plus commands such as SPOOL, SET and TTITLE are executed in the SQL*Plus program and are not passed to the database for execution. Because of this, these commands do not understand bind variables.

To use a bind variable's value as the name of a spool file:

```
-- Set a bind variable to a text string
variable mybindvar varchar2(20)
begin
  :mybindvar := 'myspoolfilename';
end;

-- Transfer the value from the bind variable to the substitution
variable
column mc new_value mysubvar noprint
select :mybindvar mc from dual;

-- Use the substitution variable
spool &mysubvar..txt
select * from employees;

spool off
```

6.3.10.11 Passing Parameters to SQL*Plus Substitution Variables

You can pass parameters on the command line to a SQL*Plus script:

```
sqlplus hr/my_password @myscript.html employees "De Haan"
```

They can be referenced in the script using "&1" and "&2". For example, myscript.sql could be:

```
set verify off
select employee_id from &1 where last_name = '&2';
```

Here the "SET VERIFY OFF" command stops SQL*Plus from echoing the SQL statement before and after the variables are substituted. The query returns the employee identifier for the employee "De Haan" from the "employees" table.

Parameters can also be passed to scripts called within SQL*Plus:

```
SQL> @myscript.sql employees "De Haan"
```

6.3.10.12 Passing Operating System Variables to SQL*Plus

You can pass an operating system variable to a SQL*Plus script as a command line parameter. For example, on UNIX:

```
sqlplus hr/my_password @myscript.sql $USER
```

or in a Windows command window:

```
sqlplus hr/my_password @myscript.sql %USERNAME%
```

The script myscript.sql could reference the substitution variable "&1" to see the passed name.

6.3.10.13 Passing a Value to a PL/SQL Procedure From the Command Line

If you create a procedure "myproc":

```
create or replace procedure myproc (p1 in number) as
begin
  dbms_output.put_line('The number is '||p1);
end;
/
```

and myscript.sql contains:

```
begin
  myproc(&1);
end;
/
```

then calling:

```
sqlplus hr/my_password @myscript.sql 88
```

executes the script as if it is:

```
begin
  myproc(88);
end;
/
```

This method does not work if the parameter "p1" to "myproc" is "IN OUT". The variable reference is pre-processed and is effectively a hardcoded value which cannot contain an

OUT value. To get around this, you can assign the substitution variable to a bind variable. The script myscript.sql becomes:

```
variable mybindvar number
begin
  :mybindvar := &1;
  myproc(:mybindvar);
end;
/
```

6.3.10.14 Allowing Script Parameters to be Optional and Have a Default Value

The goal is to create a script that accepts an optional parameter. If a parameter is passed from the command line, then its value should be used. However, if there is no parameter, then SQL*Plus should ask for a value with a customized prompt. Perhaps the closest solution is with a PROMPT/DEFINE sequence. If myscript.sql is:

```
-- Name: myscript.sql
prompt Enter a value for PAGESIZE
set termout off
define mypar = &1
set termout on
prompt Setting PAGESIZE to &mypar
set pagesize &mypar
select last_name from employees where rownum < 20;
exit
```

you can call the script with or without a parameter. If you enter "12" at the prompt your screen looks like:

```
$ sqlplus hr/my_password @myscript.sql
SQL*Plus: Release 9.2.0.3.0 - Production on Wed Mar 5 15:19:40 2003
. . .
Enter a value for PAGESIZE      12
Setting PAGESIZE to 12

LAST_NAME
-------------------------
King
Kochhar
De Haan
. . .
```

or if you call it with a parameter "8":

```
$ sqlplus hr/my_password @myscript.sql 8
SQL*Plus: Release 9.2.0.3.0 - Production on Wed Mar 5 15:20:38 2003
. . .
Enter a value for PAGESIZE
Setting PAGESIZE to 8

LAST_NAME
```

```
------------------------
King
Kochhar
De Haan
. . .
```

Note when you pass a parameter, the PROMPT text is still displayed, but you do not enter a value. The PROMPT command is the SQL*Plus "echo" or "print" statement. (It does not read input). The only occurrence of "&1" should be where "mypar" is defined. All other references to the parameter should use "&mypar" or "&&mypar".

6.3.10.15 Using a Variable for the SQL*Plus Return Status

To use the value of a substitution variable called "myv" as the SQL*Plus return status, use:

```
EXIT myv
```

No ampersand (&) prefix is required before the substitution variable name.

A numeric bind variable requires a colon (:) prefix:

```
EXIT :mybv
```

6.3.10.16 Putting the Username and Database in the Prompt

In SQL*Plus 10g, add the following to your glogin.sql or login.sql:

```
set sqlprompt "_user'@'_connect_identifier:SQL> "
```

For customized prompts that query the database, ensure that you explicitly DEFINE any referenced substitution variables. Glogin.sql and login.sql can get run when there is no database connection. Defining variables prevents the user being prompted for values when the query fails and the variables do not get defined by it:

```
set termout off
define myv = 'Not connected'
column myc new_value myv
select user||'@'||global_name myc from global_name;
set sqlprompt '&myv:SQL> '
set termout on
```

SQL*Plus 9.2 and earlier do not re-execute glogin.sql and login.sql after CONNECT commands. Also, variables in the SQLPROMPT are not dynamically substituted. It is possible to use the query script given above, but note that the prompt will only be valid for the original connection.

6.4 System Variables Influencing Substitution Variables

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

System Variable	Affect on Substitution Variables
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 (.).
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 NUMFORMAT	Sets the default format for displaying numbers, including numeric substitution variables.
SET NUMWIDTH	Sets the default width for displaying numbers, including numeric substitution variables.
SET SQLPROMPT	Sets the SQL*Plus command prompt.
SET VERIFY ON	Lists each line of the script before and after substitution.

See SET for more information about system variables.

6.4.1 System Variables in Titles and EXIT

There is a special syntax to reference system variables in TTITLE, BTITLE, REPHEADER, REPFOOTER, and EXIT commands. The name of each special variable is the same as the SHOW option prefixed with "SQL.".

The special variables that can be referenced include:

- SQL.PNO - page number
- SQL.LNO - line number
- SQL.USER - current username
- SQL.RELEASE - SQL*Plus version
- SQL.SQLCODE - last Oracle "ORA" error number

For example:

```
SQL> ttitle left 'Salary Report. Page: ' sql.pno
SQL> select salary from employees;
SQL> exit sql.sqlcode
```

System variables of numeric type, such as SQL.SQLCODE, are formatted using the same rules as numeric substitution variables.

The variables cannot be prefixed with an "&".

These variables are not substitution variables. The DEFINE command does not show them. They cannot be referenced in general commands. The system variables are not

affected if you create substitution variables with the same name. For example, SQL.USER is not affected if you create a substitution variable called USER. The system variable SQL.RELEASE is not affected if the predefined substitution variable _O_RELEASE is changed.

6.5 Passing Parameters through the START Command

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

You do this by placing an ampersand (&) followed by a numeral in the script in place of a substitution variable. Each time you run this script, 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 script called MYFILE:

```
SELECT * FROM EMP_DETAILS_VIEW
WHERE JOB_ID='&1'
AND SALARY='&2';
```

In the following START command, SQL*Plus would substitute PU_CLERK for &1 and 3100 for &2 in the script MYFILE:

```
START MYFILE PU_CLERK 3100
```

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

```
1   COLUMN LAST_NAME HEADING 'LAST NAME'
2   COLUMN SALARY  HEADING 'MONTHLY SALARY' FORMAT $99,999
3   COLUMN COMMISSION_PCT HEADING 'COMMISSION %' FORMAT 90.90
4   SELECT LAST_NAME, SALARY, COMMISSION_PCT
5   FROM EMP_DETAILS_VIEW
6* WHERE JOB_ID='SA_MAN'


6


6* WHERE JOB_ID='SA_MAN'


CHANGE /SA_MAN/&1


6* WHERE JOB_ID='&1'


SAVE ONEJOB


Created file ONEJOB
```

Now run the command with the parameter SA_MAN:

```
START ONEJOB SA_MAN
```

ORACLE®

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_ID='&1'
new   3: WHERE JOB_ID='SA_MAN'

LAST NAME                   MONTHLY SALARY COMMISSION %
-------------------------- -------------- ------------
Russell                            $14,000         0.40
Partners                           $13,500         0.30
Errazuriz                          $12,000         0.30
Cambrault                          $11,000         0.30
Zlotkey                            $10,500         0.20
```

You can use many parameters in a script. Within a script, you can refer to each parameter many times, and you can include the parameters in any order.

While you cannot use parameters when you run a command with RUN or slash (/), you could use substitution variables instead.

Before continuing, return the columns to their original heading by entering the following command:

```
CLEAR COLUMN
```

Example 6-3   Passing Parameters through START

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

```
GET SALES
```

6.5.1 Script Parameters

Parameters can be passed to SQL*Plus scripts. For example, from the command line:

```
sqlplus hr/my_password @myscript.sql King
```

You can also pass parameters when calling a SQL*Plus script from within a SQL*Plus session, for example:

```
SQL> @myscript.sql King
```

Script parameters become defined substitution variables. The variable name for the first parameter is "1", the second is "2", etc. The effect is the same as starting SQL*Plus and typing:

```
SQL> define 1 = King
SQL> @myscript.sql
```

Commands in myscript.sql can reference "&1" to get the value "King". A DEFINE command shows the parameter variable:

```
SQL> define 1
DEFINE 1        = "King" (CHAR)
```

Script parameter variables have type CHAR, similar to variables explicitly created with DEFINE.

Quoting parameters with single or double quotes is allowed. This lets whitespace be used within parameters. Operating systems and scripting languages that call SQL*Plus handle quotes in different ways. They may or may not pass quotes to the SQL*Plus executable. For example, in a standard Bourne shell on UNIX, quotes around parameters are stripped before the parameters are passed to SQL*Plus, and SQL*Plus never sees the quotes.

It is recommended to check how quoted parameters are handled on your operating system with your patch level of SQL*Plus. For portability between UNIX and Windows environments use double quotes around parameters containing whitespace.

SQL*Plus Releases 8.1.7, 9.2.0.3 (and other 9.x versions patched for bug 2471872) and 10.1 onwards remove an outer set of single or double quotes from parameters passed on the SQL*Plus command line. This makes SQL*Plus behave the same way on operating systems that do not themselves strip quotes as it does when the operating system strips the quotes before calling SQL*Plus.

As an example of passing parameters, when SQL*Plus 10.1 is called in the UNIX shell script:

```
#! /bin/sh
sqlplus hr/<i>my_password</i> @myscript.sql "Jack and Jill"
```

only one program parameter is defined. References in myscript.sql to "&1" are replaced with "Jack and Jill" (without quotes - because the shell script does not pass quotes to SQL*Plus).

6.6 About 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.

6.6.1 Receiving a Substitution Variable Value

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

```
Created file PROMPT1.sql
```

The TTITLE command sets the top title for your report. See About Defining Page and Report Titles and Dimensions for more information about the TTITLE command.

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

```
START PROMPT1


Enter a title of up to 30 characters
Title: Department Report
Department ReportEMPLOYEE_ID FIRST_NAME
LAST_NAME                        SALARY
----------- --------------------- ------------------------- ----------
        145 John                  Russell                        14000
        146 Karen                 Partners                       13500
        147 Alberto               Errazuriz                      12000
        148 Gerald                Cambrault                      11000
        149 Eleni                 Zlotkey                        10500
```

Before continuing, turn the TTITLE command off:

```
TTITLE OFF
```

Example 6-4   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:

```
CLEAR BUFFER
```

Next, set up a script as shown and save this file as PROMPT1:

```
PROMPT Enter a title of up to 30 characters
ACCEPT MYTITLE PROMPT 'Title: '
TTITLE LEFT MYTITLE SKIP 2
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN'

SAVE PROMPT1
```

6.6.2 Customizing Prompts for Substitution Variable

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.

```
Enter a valid employee ID
For Example 145, 206

Employee ID. :


205


old   3: WHERE EMPLOYEE_ID=&ENUMBER
new   3: WHERE EMPLOYEE_ID=      205
```

```
Department Report

FIRST_NAME           LAST_NAME                      SALARY
-------------------- ------------------------ ----------
Shelley              Higgins                        12000
```

What would happen if you typed characters instead of numbers? Since you specified NUMBER after the variable name in the ACCEPT command, SQL*Plus will not accept a non-numeric value:

Try entering characters instead of numbers to the prompt for "Employee ID.", SQL*Plus will respond with an error message and prompt you again to re-enter the correct number:

```
START PROMPT2
```

When SQL*Plus prompts you to enter an Employee ID, enter the word "one" instead of a number:

```
Enter a valid employee ID
For Example 145, 206

Employee ID. :


one


SP2-0425: "one" is not a valid number
```

Example 6-5   Using PROMPT and ACCEPT in Conjunction with Substitution Variables

As you have seen in Example 6-4, 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 script with the query that references the substitution variable. First clear the buffer with:

```
CLEAR BUFFER
```

To create such a file, enter the following:

```
INPUT
PROMPT Enter a valid employee ID
PROMPT For Example 145, 206
ACCEPT ENUMBER NUMBER PROMPT 'Employee ID. :'
SELECT FIRST_NAME, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE EMPLOYEE_ID=&ENUMBER;
```

Save this file as PROMPT2. Next, run this script. SQL*Plus prompts for the value of ENUMBER using the text you specified with PROMPT and ACCEPT:

```
START PROMPT2
```

SQL*Plus prompts you to enter an Employee ID:

6.6.3 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 press Return after reading the message, use the SQL*Plus command PAUSE. For example, you might include the following lines in a script:

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

6.6.4 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 script, using the following format:

```
CLEAR SCREEN
```

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

```
CLEAR COLUMNS
```

6.7 About Using Bind Variables

Bind variables are variables you create in SQL*Plus and then reference in PL/SQL or 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 a bind variable as an input bind variable to hold data which can then be used in PL/SQL or SQL statements to insert data into the database. You can assign a value to a newly defined variable. The value assigned in this variable can then be used in a statement.

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

6.7.1 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 the VARIABLE command for more information. (To list all bind variables created in a session, type VARIABLE without any arguments.)

6.7.2 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;
```

To change this bind variable in SQL*Plus, you must enter a PL/SQL block. For example:

```
BEGIN
 :ret_val:=4;
END;
/


PL/SQL procedure successfully completed.
```

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

6.7.3 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


    RET_VAL
----------
         4
```

This command displays a bind variable named ret_val. See PRINT for more information about displaying bind variables.

6.7.4 Executing an Input Bind

You can assign a value to a variable for input binding.

```
SQL> variable abc number=123
SQL> select :abc from dual;

      :ABC
----------
       123

SQL>


SQL> create table mytab (col1 number, col2 varchar2(10));

Table created.

SQL> var abc number=123
SQL> var xyz varchar2(10)='test'
SQL> insert into mytab values(:abc,:xyz);

1 row created.

SQL> select * from mytab;

      COL1 COL2
---------- ----------
       123 test

SQL>
```

See the VARIABLE command for more information.

6.7.5 Limitation

If the client character `NLS_LANG` environment variable is not set and the database character set is multibyte, for example, `AL32UTF32`, then PL/SQL will truncate the data when the declared bind variable is smaller than the data returned from PL/SQL.

```
VAR a VARCHAR2(1)
 BEGIN
  :a := '12';
 END;
 /
```

Print the value of the bind variable `a`:

```
PRINT a;
```

The output gets truncated. SQL*Plus displays the output as '1' instead of '12', without displaying any errors.

```
A
-------------------------------
1
```

If the client character (`NLS_LANG`) is set to the same character set as the database (that is, both `NLS_LANG` and the database character set are `AL32UTF8`), then PL/SQL will return an error.

```
setenv NLS_LANG.AL32UTF8
VAR a VARCHAR2(1)
BEGIN
 :a := '12';
END;
/
BEGIN
*
ERROR at line 1:
ORA-06502: PL/SQL: value or conversion error: character string buffer
too small
```

6.8 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 enables 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.

```
PL/SQL procedure successfully completed.
```

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

```
PRINT employee_info
```

```
EMPLOYEE_ID     SALARY
----------- ----------
        145      14000
        146      13500
        147      12000
        148      11000
        149      10500
```

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

```
Package created.
```

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

```
CREATE OR REPLACE PACKAGE BODY EmpPack AS
  PROCEDURE EmpInfoRpt (emp_cv IN OUT EmpInfoTyp) AS
  BEGIN
    OPEN emp_cv FOR SELECT EMPLOYEE_ID, SALARY
    FROM EMP_DETAILS_VIEW
    WHERE JOB_ID='SA_MAN' ;
  END;
END;
 /
```

```
Procedure created.
```

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

```
VARIABLE cv REFCURSOR
EXECUTE EmpPack.EmpInfoRpt(:cv)
```

```
PL/SQL procedure successfully completed.
```

Now print the bind variable.

```
PRINT cv
```

```
EMPLOYEE_ID     SALARY
----------- ----------
        145      14000
        146      13500
        147      12000
```

```
       148      11000
       149      10500
```

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

```
VARIABLE pcv REFCURSOR
EXECUTE EmpInfo_rpt(:pcv)


PL/SQL procedure successfully completed.


PRINT pcv


EMPLOYEE_ID     SALARY
----------- ----------
        145      14000
        146      13500
        147      12000
        148      11000
        149      10500



Function created.
```

Execute the function.

```
VARIABLE rc REFCURSOR
EXECUTE :rc := EmpInfo_fn


PL/SQL procedure successfully completed.
```

Now print the bind variable.

```
PRINT rc


EMPLOYEE_ID     SALARY
----------- ----------
        145      14000
        146      13500
        147      12000
        148      11000
        149      10500
```

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

```
EXECUTE :rc := EmpInfo_fn


PL/SQL procedure successfully completed.
```

Example 6-6   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

```
create procedure p4 as
  c1 sys_refcursor;
  c2 sys_refcursor;
begin
  open c1 for SELECT * FROM DEPT;
  dbms_sql.return_result(c1);
  open c2 for SELECT * FROM EMP;
  dbms_sql.return_result(c2);
end;
/
```

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 OPEN Statementfor information on the OPEN command and cursor variables.

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

```
BEGIN
OPEN :employee_info FOR SELECT EMPLOYEE_ID, SALARY
FROM    EMP_DETAILS_VIEW WHERE JOB_ID='SA_MAN' ;
END;
 /
```

Example 6-7   Using REFCURSOR Variables in Stored Procedures

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

```
CREATE OR REPLACE PACKAGE EmpPack AS
  TYPE EmpInfoTyp IS REF CURSOR;
  PROCEDURE EmpInfoRpt (emp_cv IN OUT EmpInfoTyp);
END EmpPack;
/
```

Example 6-8   Using REFCURSOR Variables in Stored Functions

Create a stored function containing an OPEN... FOR SELECT statement:

```
CREATE OR REPLACE FUNCTION EmpInfo_fn RETURN -
cv_types.EmpInfo IS
resultset cv_types.EmpInfoTyp;
BEGIN
OPEN resultset FOR SELECT EMPLOYEE_ID, SALARY
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN';
RETURN(resultset);
END;
/
```

6.9 Fetching Iterative Results from a SELECT inside a PL/SQL Block

SQL*Plus can iteratively fetch and format the results of a SELECT statement contained in a PL/SQL block or stored procedure. You do not need to define local REFCURSOR variables.

The results from the SELECT statements are displayed.

```
ResultSet #1


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

4 rows selected


ResultSet #2


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      10
7788  SCOTT  ANALYST   7566 05-APR-11 3000      20
7839  KING   PRESIDENT      17-NOV-81 5000      10
7844  TURNER SALESMAN  7698 08-SEP-81 1500    0 30
7876  ADAMS  CLERK     7788 09-MAY-11 1100

14 rows selected
```

Example 6-9   Creating a PL/SQL Procedure

Create a PL/SQL procedure P4 which calls two statements.

```
create procedure p4 as
    c1 sys_refcursor;
    c2 sys_refcursor;
begin
    open c1 for SELECT * FROM DEBT;
    dbms_sql.return_result(c1);
    open c2 for SELECT * FROM EMP;
    dbms_sql.return_result(c2);
end;
/
Procedure created.
```

Next, run the procedure to retrieve results iteratively from the SELECT statements in the procedure.

```
exec p4
PL/SQL procedure successfully completed.
```

7

Formatting SQL*Plus Reports

This chapter explains how to format your query results to produce a finished report. This chapter does not discuss HTML output, but covers the following topics:

- About Formatting Columns
- About Clarifying Your Report with Spacing and Summary Lines
- About Defining Page and Report Titles and Dimensions
- About 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 HR sample schema described in SQL*Plus Quick Start .

7.1 About Formatting Columns

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

7.1.1 About Changing Column Headings

When displaying column headings, you can either use the default heading or you can change it using the COLUMN command. The following sections describe how default headings are derived and how to alter them using the COLUMN command. See the COLUMN command for more details.

7.1.1.1 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.

7.1.1.2 Changing Default Headings

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

```
COLUMN column_name HEADING column_heading
```

```
LAST NAME                 MONTHLY SALARY COMMISSION
------------------------- -------------- ----------
Russell                            14000         .4
Partners                           13500         .3
Errazuriz                          12000         .3
Cambrault                          11000         .3
Zlotkey                            10500         .2
```

> **✏ 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 for more
information.)

```
LAST                           MONTHLY
NAME                            SALARY COMMISSION
-------------------------- ---------- ----------
Russell                         14000         .4
Partners                        13500         .3
Errazuriz                       12000         .3
Cambrault                       11000         .3
Zlotkey                         10500         .2


LAST                           MONTHLY
NAME                            SALARY COMMISSION
========================== ========== ==========
Russell                         14000         .4
Partners                        13500         .3
Errazuriz                       12000         .3
Cambrault                       11000         .3
Zlotkey                         10500         .2
```

Now change the underline character back to a dash:

```
SET UNDERLINE '-'
```

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

Example 7-1   Changing a Column Heading

To produce a report from EMP_DETAILS_VIEW with new headings specified for
LAST_NAME, SALARY, and COMMISSION_PCT, enter the following commands:

```
COLUMN LAST_NAME        HEADING 'LAST NAME'
COLUMN SALARY           HEADING 'MONTHLY SALARY'
COLUMN COMMISSION_PCT   HEADING COMMISSION
SELECT LAST_NAME, SALARY, COMMISSION_PCT
```

```
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN';
```

Example 7-2   Splitting a Column Heading

To give the columns SALARY and LAST_NAME the headings MONTHLY SALARY and LAST NAME respectively, and to split the new headings onto two lines, enter

```
COLUMN SALARY HEADING 'MONTHLY|SALARY'
COLUMN LAST_NAME HEADING 'LAST|NAME'
```

Now rerun the query with the slash (/) command:

```
 /
```

Example 7-3   Setting the Underline Character

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

```
SET UNDERLINE =
/
```

7.1.2 About 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. Later sections describe the default display and how you can alter it with the COLUMN command. The format model will stay in effect until you enter a new one, reset the column's format with

```
COLUMN column_name CLEAR
```

or exit from SQL*Plus.

7.1.2.1 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 if possible, or displays hashes if the number is too large.

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 9s to represent digits.

7.1.2.2 Changing the Default Display

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

```
COLUMN column_name FORMAT model
```

Use format models to add commas, dollar signs, angle brackets (around negative values), and 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 7-4). See COLUMN for a complete list of format models and further details.

```
LAST                    MONTHLY
NAME                     SALARY COMMISSION
------------------------ -------- ----------
Russell                  $14,000        .4
Partners                 $13,500        .3
Errazuriz                $12,000        .3
Cambrault                $11,000        .3
Zlotkey                  $10,500        .2
```

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

Example 7-4   Formatting a NUMBER Column

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

```
COLUMN SALARY FORMAT $99,990
```

Now rerun the current query:

```
/
```

7.1.3 About Formatting Datatypes

When displaying datatypes, you can either accept the SQL*Plus default display width or you can change it using the COLUMN command. The format model will stay in effect until you enter a new one, reset the column's format with

```
COLUMN column_name CLEAR
```

or exit from SQL*Plus. Datatypes, in this manual, include the following types:

- CHAR
- NCHAR
- VARCHAR2 (VARCHAR)
- NVARCHAR2 (NCHAR VARYING)
- DATE
- LONG
- BLOB

  BFILE
- CLOB
- NCLOB

- XMLType
- JSON

7.1.3.1 Default Display

The default width of datatype columns is the width of the column in the database. The column width of a LONG, BLOB, BFILE, CLOB, NCLOB or XMLType defaults to the value of SET LONGCHUNKSIZE or SET LONG, whichever is the smaller.

The default width and format of unformatted DATE columns in SQL*Plus is determined by the database NLS_DATE_FORMAT parameter. Otherwise, the default format width is A9. See the FORMAT clause of the COLUMN command for more information on formatting DATE columns.

Left justification is the default for datatypes.

7.1.3.2 Changing the Default Display

You can change the displayed width of a datatype or DATE, 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. See the COLUMN command for more details.

```
LAST   MONTHLY
NAME    SALARY COMMISSION
----  -------- ----------
Russ  $14,000         .4
ell

Part  $13,500         .3
ners

Erra  $12,000         .3
zuri
z


LAST   MONTHLY
NAME    SALARY COMMISSION
----  -------- ----------
Camb  $11,000         .3
raul
t

Zlot  $10,500         .2
key
```

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 7-5. 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 the COLUMN command for more information on these clauses. You will use the WORD_WRAPPED clause of COLUMN later in this chapter.

NCLOB, BLOB, BFILE or multibyte CLOB columns cannot be formatted with the WORD_WRAPPED option. If you format an NCLOB, BLOB, BFILE or multibyte CLOB column with COLUMN WORD_WRAPPED, the column data behaves as though COLUMN WRAPPED was applied instead.

> **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:

```
COLUMN LAST_NAME FORMAT A10
```

```
Building
-------------------
Owned
```

For more information about the createXML, extract, text and getStringVal functions, and about creating and manipulating XMLType data, see *Oracle Database PL/SQL Packages and Types Reference*.

Example 7-5   Formatting a Character Column

To set the width of the column LAST_NAME to four characters and rerun the current query, enter

```
COLUMN LAST_NAME FORMAT A4
/
```

Example 7-6   Formatting an XMLType Column

Before illustrating how to format an XMLType column, you must create a table with an XMLType column definition, and insert some data into the table. You can create an XMLType column like any other user-defined column. To create a table containing an XMLType column, enter

```
CREATE TABLE warehouses (
  warehouse_id  NUMBER(3),
  warehouse_spec  SYS.XMLTYPE,
  warehouse_name  VARCHAR2 (35),
  location_id  NUMBER(4));
```

To insert a new record containing warehouse_id and warehouse_spec values into the new warehouses table, enter

```
INSERT into warehouses (warehouse_id, warehouse_spec)
  VALUES (100, sys.XMLTYPE.createXML(
  '<Warehouse whNo="100">
    <Building>Owned</Building>
  </Warehouse>'));
```

To set the XMLType column width to 20 characters and then select the XMLType column, enter

```
COLUMN Building FORMAT A20
SELECT
  w.warehouse_spec.extract('/Warehouse/Building/text()').getStringVal()
  "Building"
  FROM warehouses w;
```

7.1.4 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.

```
LAST        MONTHLY
NAME         SALARY    BONUS
---------- -------- --------
Russell    $14,000       $0
Partners   $13,500       $0
Errazuriz  $12,000       $0
Cambrault  $11,000       $0
Zlotkey    $10,500       $0
```

Example 7-7   Copying a Column's Display Attributes

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

```
COLUMN COMMISSION_PCT LIKE SALARY HEADING BONUS
```

Rerun the query:

```
/
```

7.1.5 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:

```
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:

```
COLUMN column_name CLEAR
```

```
columns cleared
```

Example 7-8   Resetting Column Display Attributes to their Defaults

To reset all column display attributes to their default values, enter:

```
CLEAR COLUMNS
```

7.1.6 About 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
```

OFF 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
```

7.1.7 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 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:

```
COLUMN column_name WORD_WRAPPED
```

```
LAST_NAME                JOB_TITLE            CITY
------------------------ -------------------- --------
King                     President            Seattle
Kochhar                  Administration Vice  Seattle
                         President
--------------------------------------------------------
De Haan                  Administration Vice  Seattle
                         President
--------------------------------------------------------
```

```
Russell                    Sales Manager        Oxford
Partners                   Sales Manager        Oxford
Hartstein                  Marketing Manager    Toronto

6 rows selected.
```

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:

```
SET RECSEP OFF
```

Example 7-9   Printing a Line of Characters after Wrapped Column Values

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

```
SET RECSEP WRAPPED
SET RECSEPCHAR "-"
```

Finally, enter the following query:

```
SELECT LAST_NAME, JOB_TITLE, CITY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
```

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

```
COLUMN JOB_TITLE FORMAT A20 WORD_WRAPPED
```

Run the query:

```
/
```

7.2 About 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 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.

```
SELECT DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY > 12000
ORDER BY DEPARTMENT_ID;

DEPARTMENT_ID LAST_NAME                        SALARY
------------- ------------------------- ----------
           20 Hartstein                         13000
```

```
         80 Russell                         14000
         80 Partners                        13500
         90 King                            24000
         90 Kochhar                         17000
         90 De Haan                         17000

6 rows selected.
```

To make this report more useful, you would use BREAK to establish DEPARTMENT_ID as the break column. Through BREAK you could suppress duplicate values in DEPARTMENT_ID 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 salary for each department and for all departments. You could also print summary lines containing the average, maximum, minimum, standard deviation, variance, or row count.

7.2.1 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, breaks occur every time the column value changes.

```
DEPARTMENT_ID LAST_NAME                     SALARY
------------- ------------------------- ----------
           20 Hartstein                   13000
           80 Russell                     14000
              Partners                    13500
           90 King                        24000
              Kochhar                     17000
              De Haan                     17000

6 rows selected.
```

Example 7-10   Suppressing Duplicate Values in a Break Column

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

```
BREAK ON DEPARTMENT_ID;
```

For the following query (which is the current query stored in the buffer):

```
SELECT DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
```

```
WHERE SALARY > 12000
ORDER BY DEPARTMENT_ID;
```

7.2.2 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
```

```
DEPARTMENT_ID LAST_NAME                     SALARY
------------- ------------------------ ----------
           20 Hartstein                     13000

           80 Russell                       14000
              Partners                      13500

           90 King                          24000
              Kochhar                       17000
              De Haan                       17000
```

```
6 rows selected.
```

Example 7-11   Inserting Space when a Break Column's Value Changes

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

```
BREAK ON DEPARTMENT_ID SKIP 1
```

Now rerun the query:

```
/
```

7.2.3 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 character to be generated unless you have also specified NEWPAGE 0.

7.2.4 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.

```
                              Page: 1
DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------ ----------
           20 MK_MAN     Hartstein                     13000

                              Page: 2
DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------ ----------
           80 SA_MAN     Russell                       14000
                         Partners                      13500

                              Page: 3
DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------ ----------
           90 AD_PRES    King                          24000

              AD_VP      Kochhar                       17000
                         De Haan                       17000
```

6 rows selected.

Example 7-12   Combining Spacing Techniques

Type the following:

```
SELECT DEPARTMENT_ID, JOB_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000
ORDER BY DEPARTMENT_ID, JOB_ID;
```

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

```
BREAK ON DEPARTMENT_ID SKIP PAGE ON JOB_ID SKIP 1
```

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

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

Run the new query to see the results:

7.2.5 Listing and Removing Break Definitions

Before continuing, turn off the top title display without changing its definition:

```
TTITLE OFF
```

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 script to ensure that previously entered BREAK commands will not affect queries you run in a given file.

7.2.6 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. For more information see the COMPUTE command.

The following table lists compute functions and their effects

Table 7-1   Compute Functions

Function...	Computes the...
SUM	Sum of the values in the column.
MINIMUM	Minimum value in the column.
MAXIMUM	Maximum value in the column.
AVG	Average of the values in the column.
STD	Standard deviation of the values in the column.

Table 7-1   (Cont.) Compute Functions

Function...	Computes the...
VARIANCE	Variance of the values in the column.
COUNT	Number of non-null values in the column.
NUMBER	Number of rows in the column.

The function you specify in the COMPUTE command applies to all columns you enter after OF 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 for more details. If you use the NOPRINT option for the column on which the COMPUTE is being performed, the COMPUTE result is also suppressed.

```
break on DEPARTMENT_ID page  nodup
         on JOB_ID skip 1 nodup
```

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

```
COMPUTE SUM OF SALARY ON DEPARTMENT_ID
/

DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------- ----------
           20 MK_MAN     Hartstein                      13000
************* **********                           ----------
sum                                                    13000

DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------- ----------
           80 SA_MAN     Russell                        14000
                         Partners                       13500

************* **********                           ----------
sum                                                    27500

DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------- ----------
           90 AD_PRES    King                           24000

              AD_VP      Kochhar                        17000
```

```
                             De Haan                            17000

************* **********                                    ----------
sum                                                             58000

6 rows selected.
```

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

```
COLUMN DUMMY NOPRINT;
COMPUTE SUM OF SALARY ON DUMMY;
BREAK ON DUMMY SKIP 1;
SELECT DEPARTMENT_ID DUMMY,DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000
ORDER BY DEPARTMENT_ID;


DEPARTMENT_ID LAST_NAME                    SALARY
------------- ------------------------- ----------
           20 Hartstein                    13000
                                        ----------
                                            13000

           80 Russell                      14000
           80 Partners                     13500
                                        ----------
                                            27500

           90 King                         24000
           90 Kochhar                      17000
           90 De Haan                      17000
                                        ----------
                                            58000

6 rows selected.
```

To compute the salaries just at the end of the report:

```
COLUMN DUMMY NOPRINT;
COMPUTE SUM OF SALARY ON DUMMY;
BREAK ON DUMMY;
SELECT NULL DUMMY,DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000
ORDER BY DEPARTMENT_ID;


DEPARTMENT_ID LAST_NAME                    SALARY
------------- ------------------------- ----------
           20 Hartstein                    13000
           80 Russell                      14000
           80 Partners                     13500
           90 King                         24000
           90 Kochhar                      17000
           90 De Haan                      17000
                                        ----------
```

```
                                                  98500
```

```
6 rows selected.
```

When you establish the format of a NUMBER column, you must allow for the size of the sums included in the report.

Example 7-13   Computing and Printing Subtotals

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

```
BREAK
```

which displays current BREAK definitions:

7.2.7 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
```

```
LAST_NAME                     SALARY
------------------------- ----------
Russell                        14000
Partners                       13500
Errazuriz                      12000
Cambrault                      11000
Zlotkey                        10500
                          ----------
TOTAL                          61000
```

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
```

Example 7-14   Computing and Printing a Grand Total

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

```
BREAK ON REPORT
COMPUTE SUM LABEL TOTAL OF SALARY ON REPORT
```

Next, enter and run a new query:

```
SELECT LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN';
```

7.2.8 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.

```
1* SELECT LAST_NAME, SALARY


APPEND , COMMISSION_PCT;
```

Finally, run the revised query to see the results:

```
/


LAST_NAME                      SALARY COMMISSION_PCT
-------------------------- ---------- --------------
Russell                         14000            .4
Partners                        13500            .3
Errazuriz                       12000            .3
Cambrault                       11000            .3
Zlotkey                         10500            .2
                           ---------- --------------
sum                             61000           1.5
```

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 OF and before ON, COMPUTE calculates and prints values for each column you specify.

```
DEPARTMENT_ID LAST_NAME                      SALARY
------------- -------------------------- ----------
           30 Colmenares                       2500
              Himuro                           2600
              Tobias                           2800
              Baida                            2900
              Khoo                             3100
              Raphaely                        11000
*************                            ----------
avg                                            4150
sum                                           24900

6 rows selected.
```

Example 7-15   Computing the Same Type of Summary Value on Different Columns

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

```
COMPUTE SUM OF SALARY COMMISSION_PCT ON REPORT
```

You do not have to enter a BREAK command; the BREAK you entered in
Example 7-14 is still in effect. Now, change the first line of the select query to include
COMMISSION_PCT:

```
1
```

Example 7-16   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:

```
BREAK ON DEPARTMENT_ID
COMPUTE AVG SUM OF SALARY ON DEPARTMENT_ID
```

Now, enter and run the following query:

```
SELECT DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE DEPARTMENT_ID = 30
ORDER BY DEPARTMENT_ID, SALARY;
```

7.2.9 Listing and Removing COMPUTE Definitions

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

```
COMPUTE
```

```
breaks cleared
```

```
CLEAR COMPUTES
```

```
computes cleared
```

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

Example 7-17   Removing COMPUTE Definitions

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

```
CLEAR BREAKS
```

7.3 About Defining Page and Report Titles and Dimensions

The word page refers to a screen full 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.

7.3.1 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 ...
```

For descriptions of all TTITLE, BTITLE, REPHEADER and REPFOOTER clauses, see the TTITLE command and the REPHEADER command.

```
                 ACME  SALES  DEPARTMENT  PERSONNEL  REPORT
DEPARTMENT_ID LAST_NAME                        SALARY
------------- ------------------------- ----------
           30 Colmenares                         2500
           30 Himuro                             2600
           30 Tobias                             2800
           30 Baida                              2900
           30 Khoo                               3100
           30 Raphaely                          11000



                    COMPANY  CONFIDENTIAL

6 rows selected.


                ACME  SALES  DEPARTMENT  PERSONNEL  REPORT
                          PERFECT  WIDGETS


                    COMPANY  CONFIDENTIAL

                 ACME  SALES  DEPARTMENT  PERSONNEL  REPORT
DEPARTMENT_ID LAST_NAME                        SALARY
------------- ------------------------- ----------
           30 Colmenares                         2500
           30 Himuro                             2600
           30 Tobias                             2800
           30 Baida                              2900
           30 Khoo                               3100
           30 Raphaely                          11000


                    COMPANY  CONFIDENTIAL
```

```
6 rows selected.
```

To suppress the report header without changing its definition, enter

```
REPHEADER OFF
```

Example 7-18   Placing a Top and Bottom Title on a Page

To put titles at the top and bottom of each page of a report, enter

```
TTITLE CENTER -
"ACME SALES DEPARTMENT PERSONNEL REPORT"
BTITLE CENTER "COMPANY CONFIDENTIAL"
```

Now run the current query:

```
/
```

Example 7-19   Placing a Header on a Report

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

```
REPHEADER PAGE CENTER 'PERFECT WIDGETS'
```

Now run the current query:

```
/
```

which displays the following two pages of output, with the new REPHEADER displayed on the first page:

7.3.1.1 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 line size 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.

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

DEPARTMENT_ID LAST_NAME                      SALARY
------------- ------------------------- ----------
           30 Colmenares                      2500
           30 Himuro                          2600
           30 Tobias                          2800
           30 Baida                           2900
           30 Khoo                            3100
           30 Raphaely                       11000
                  COMPANY CONFIDENTIAL
```

```
6 rows selected.
```

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'
```

Example 7-20   Positioning Title Elements

To redisplay the personnel report with a repositioned top title, enter the following commands:

```
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
SET LINESIZE 60
/
```

7.3.1.2 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.

```
ACME WIDGET
     SALES DEPARTMENT PERSONNEL REPORT


DEPARTMENT_ID LAST_NAME                     SALARY
------------- ------------------------- ----------
          30 Colmenares                      2500
          30 Himuro                          2600
          30 Tobias                          2800
          30 Baida                           2900
          30 Khoo                            3100
          30 Raphaely                       11000


             COMPANY CONFIDENTIAL

6 rows selected.
```

Example 7-21   Indenting a Title Element

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

```
TTITLE LEFT 'ACME WIDGET' SKIP 1 -
COL 6 'SALES DEPARTMENT PERSONNEL REPORT' SKIP 2
```

Now rerun the current query to see the results:

```
/
```

7.3.1.3 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 substitution variable:

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

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

```
TTITLE CENTER LINE1 SKIP 1 CENTER LINE2 SKIP 1 -
CENTER LINE3
```

7.3.2 Displaying 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). See TTITLE for a list of system-maintained values you can display in titles.

```
ACMEWIDGET                                    PAGE:          1

DEPARTMENT_ID LAST_NAME                    SALARY
------------- ------------------------- ----------
           30 Colmenares                      2500
           30 Himuro                          2600
           30 Tobias                          2800
           30 Baida                           2900
           30 Khoo                            3100
           30 Raphaely                       11000


              COMPANY CONFIDENTIAL

6 rows selected.
```

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

```
ACME WIDGET                                   'PAGE:'   1

DEPARTMENT_ID LAST_NAME                    SALARY
------------- ------------------------- ----------
```

```
          30 Colmenares                      2500
          30 Himuro                          2600
          30 Tobias                          2800
          30 Baida                           2900
          30 Khoo                            3100
          30 Raphaely                       11000



                       COMPANY CONFIDENTIAL

6 rows selected.
```

Example 7-22   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:

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

Now rerun the current query:

```
/
```

Example 7-23   Formatting a System-Maintained Value in a Title

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

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

Now rerun the query:

```
/
```

7.3.3 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
```

7.3.4 Displaying Column Values in Titles

You may wish to create a primary/detail report that displays a changing primary column value at the top of each page with the detail query results for that value underneath. 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 primary column in an ORDER BY clause and in a BREAK command using the SKIP PAGE clause.

```
Manager:        101

DEPARTMENT_ID LAST_NAME                       SALARY
------------- ------------------------- ----------
           10 Whalen                          4400
           40 Mavris                          6500
           70 Baer                           10000
          100 Greenberg                      12000
          110 Higgins                        12000

Manager:        201

DEPARTMENT_ID LAST_NAME                       SALARY
------------- ------------------------- ----------
           20 Fay                             6000

6 rows selected.
```

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 primary column. Therefore, if you simply referenced the NEW_VALUE of the primary column, you would get the value for the next set of details. OLD_VALUE remembers the value of the primary column that was in effect before the page break began.

Example 7-24   Creating a Primary/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:

```
COLUMN MANAGER_ID NEW_VALUE MGRVAR NOPRINT
```

Because you will only 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 MANAGER_ID.

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:

```
TTITLE LEFT 'Manager: ' MGRVAR SKIP 2
BREAK ON MANAGER_ID SKIP PAGE
BTITLE OFF
```

Finally, enter and run the following query:

```
SELECT MANAGER_ID, DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE MANAGER_ID IN (101, 201)
ORDER BY MANAGER_ID, DEPARTMENT_ID;
```

7.3.5 About 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.

You can reference the predefined substitution variable _DATE to display the current date in a title as you would any other variable.

The date format model you include in your LOGIN file or in your SELECT statement determines the format in which SQL*Plus displays the date. See your *Oracle Database SQL Language Reference* for more information on date format models. See Modifying Your LOGIN File for more information about the LOGIN file.

You can also enter these commands interactively. See COLUMN for more information.

7.3.6 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 underneath:

- number of lines before the top title: 1
- number of lines per page, from the top title to the bottom of the page: 14
- 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.

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

```
SET LINESIZE number_of_characters
```

```
                    ACME WIDGET PERSONNEL REPORT
                           01-JAN-2001


                 FIRST                LAST                     MONTHLY
DEPARTMENT_ID NAME                 NAME                         SALARY
------------- -------------------- ------------------------- --------
           90 Steven               King                       $24,000
           90 Neena                Kochhar                    $17,000
           90 Lex                  De Haan                    $17,000
           80 John                 Russell                    $14,000
           80 Karen                Partners                   $13,500
           20 Michael              Hartstein                  $13,000

6 rows selected.
```

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

```
SET PAGESIZE 14
SET NEWPAGE 1
SET LINESIZE 80
```

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

```
SHOW PAGESIZE
SHOW NEWPAGE
SHOW LINESIZE
```

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

Example 7-25   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 line size to 70, enter the following commands:

```
SET PAGESIZE 66
SET NEWPAGE 0
SET LINESIZE 70
```

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

```
TTITLE CENTER 'ACME WIDGET PERSONNEL REPORT' SKIP 1 -
CENTER '01-JAN-2001' SKIP 2
```

Now run the following query:

```
COLUMN FIRST_NAME HEADING 'FIRST|NAME';
COLUMN LAST_NAME HEADING 'LAST|NAME';
COLUMN SALARY  HEADING 'MONTHLY|SALARY' FORMAT $99,999;
SELECT DEPARTMENT_ID, FIRST_NAME, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
```

7.4 About 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, email, 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 operating system; on most hosts it is LST or LIS. The extension is not appended when you spool to system generated files such as /dev/null and /dev/stderr. See the platform-specific Oracle documentation 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
```

7.4.1 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 Oracle Net, you need to create a flat file for use with SQL*Loader when moving data from Oracle9*i* to Oracle Database 10*g*.

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 VERIFY OFF
SET HEADING OFF
SET MARKUP HTML OFF SPOOL 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.

7.4.2 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.

7.4.3 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 computer's standard (default) printer. SPOOL OUT does not delete the spool file after printing.

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

EMPLOYEE REPORT                                               PAGE: 1

DEPARTMENT LAST NAME                    MONTHLY SALARY
---------- ------------------------- ---------------
        20 Hartstein                        $13,000
**********                               --------------
sum                                         $13,000

        80 Russell                          $14,000
           Partners                         $13,500
**********                               --------------
sum                                         $27,500

        90 King                             $24,000
           Kochhar                          $17,000
           De Haan                          $17,000
**********                               --------------
sum                                         $58,000

                                         --------------
sum                                         $98,500
                       COMPANY CONFIDENTIAL

6 rows selected.
```

Example 7-26   Sending Query Results to a Printer

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

First, use EDIT to create the script with your operating system text editor.

```
EDIT EMPRPT
```

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

```
SPOOL TEMP
CLEAR COLUMNS
CLEAR BREAKS
CLEAR COMPUTES

COLUMN DEPARTMENT_ID HEADING DEPARTMENT
COLUMN LAST_NAME HEADING 'LAST NAME'
```

```
COLUMN SALARY HEADING 'MONTHLY SALARY' FORMAT $99,999

BREAK ON DEPARTMENT_ID SKIP 1 ON REPORT
COMPUTE SUM OF SALARY ON DEPARTMENT_ID
COMPUTE SUM OF SALARY ON REPORT

SET PAGESIZE 24
SET NEWPAGE 0
SET LINESIZE 70

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 DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000
ORDER BY DEPARTMENT_ID;

SPOOL OFF
```

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 and close the file in your text editor (you will automatically return to SQL*Plus). Now, run the script EMPRPT:

```
@EMPRPT
```

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

8

Generating Reports from SQL*Plus

This chapter explains how to generate a HTML and CSV reports containing your query results. This chapter covers the following topics:

- About Creating Reports using Command-line SQL*Plus

8.1 About Creating Reports using Command-line SQL*Plus

In addition to plain text output, the SQL*Plus command-line interface enables you to generate either a complete web page, HTML output which can be embedded in a web page, or data in CSV format. You can use SQLPLUS -MARKUP "HTML ON" or SET MARKUP HTML ON SPOOL ON to produce complete HTML pages automatically encapsulated with <HTML> and <BODY> tags. You can use SQLPLUS -MARKUP "CSV ON" or SET MARKUP CSV ON to produce reports in CSV format.

By default, data retrieved with MARKUP HTML ON is output in HTML, though you can optionally direct output to the HTML <PRE> tag so that it displays in a web browser exactly as it appears in SQL*Plus. See the SQLPLUS MARKUP Options and the SET MARKUP command for more information about these commands.

SQLPLUS -MARKUP "HTML ON" is useful when embedding SQL*Plus in program scripts. On starting, it outputs the HTML and BODY tags before executing any commands. All subsequent output is in HTML until SQL*Plus terminates.

The -SILENT and -RESTRICT command-line options may be effectively used with -MARKUP to suppress the display of SQL*Plus prompt and banner information, and to restrict the use of some commands.

SET MARKUP HTML ON SPOOL ON generates an HTML page for each subsequently spooled file. The HTML tags in a spool file are closed when SPOOL OFF is executed or SQL*Plus exits.

You can use SET MARKUP HTML ON SPOOL OFF to generate HTML output suitable for embedding in an existing web page. HTML output generated this way has no <HTML> or <BODY> tags.

You can enable CSV markup while logging into a user session, by using the -M[ARKUP] CSV ON option at the SQL*Plus command line. For more information, see SQL*Plus Program Syntax. While logged in to a user session, you can enable CSV markup by using the SET MARKUP CSV ON command.

You can specify the delimiter character by using the DELIMITER option. You can also output text without quotes by using QUOTE OFF.

You can suppress display of data returned by a query by using the ONLY option of the SET FEEDBACK command. The number of rows selected and returned by the query is displayed.

8.1.1 Creating HTML Reports

During a SQL*Plus session, use the SET MARKUP command interactively to write HTML to a spool file. You can view the output in a web browser.

SET MARKUP HTML ON SPOOL ON only specifies that SQL*Plus output will be HTML encoded, it does not create or begin writing to an output file. You must use the SQL*Plus SPOOL command to start generation of a spool file. This file then has HTML tags including <HTML> and </HTML>.

When creating a HTML file, it is important and convenient to specify a .html or .htm file extension which are standard file extensions for HTML files. This enables you to easily identify the type of your output files, and also enables web browsers to identify and correctly display your HTML files. If no extension is specified, the default SQL*Plus file extension is used.

You use SPOOL OFF or EXIT to append final HTML tags to the spool file and then close it. If you enter another SPOOL filename command, the current spool file is closed as for SPOOL OFF or EXIT, and a new HTML spool file with the specified name is created.

You can use the SET MARKUP command to enable or disable HTML output as required.

```
SQL> SELECT '<A HREF="http://oracle.com/'||DEPARTMENT_NAME||'.html">'||DEPARTMENT_NAME||'</A>'
DEPARTMENT_NAME, CITY
2 FROM EMP_DETAILS_VIEW
3 WHERE SALARY>12000;
```

DEPARTMENT	CITY
Executive	Seattle
Executive	Seattle
Executive	Seattle
Sales	Oxford
Sales	Oxford
Marketing	Toronto

```
6 rows selected.

SQL> SPOOL OFF
```

In this example, the prompts and query text have not been suppressed. Depending on how you invoke a script, you can use SET ECHO OFF or command-line -SILENT options to do this.

The SQL*Plus commands in this example contain several items of usage worth noting:

- The hyphen used to continue lines in long SQL*Plus commands.
- The TABLE option to set table WIDTH and BORDER attributes.
- The COLUMN command to set ENTMAP OFF for the DEPARTMENT_NAME column to enable the correct formation of HTML hyperlinks. This makes sure that any HTML special characters such as quotes and angle brackets are not replaced by their equivalent entities, ", &, < and >.
- The use of quotes and concatenation characters in the SELECT statement to create hyperlinks by concatenating string and variable elements.

View the report.html source in your web browser, or in a text editor to see that the table cells for the Department column contain fully formed hyperlinks as shown:

```
<html>
<head>
<TITLE>Department Report</TITLE>  <STYLE type="text/css">
<!-- BODY {background: #FFFFC6} -->  </STYLE>
<meta name="generator" content="SQL*Plus 10.2.0.1">
</head>
<body TEXT="#FF00Ff">
SQL&gt; SELECT '&lt;A HREF=&quot;http://oracle.com/'
||DEPARTMENT_NAME||'.html&quot;&gt;'||DEPARTMENT_NAME
||'&lt;/A&gt;' DEPARTMENT_NAME, CITY
<br>
  2  FROM EMP_DETAILS_VIEW
<br>
  3* WHERE SALARY&gt;12000
<br>
<p>
<table WIDTH="90%" BORDER="5">
<tr><th>DEPARTMENT</th><th>CITY</th></tr>
<tr><td><A HREF="http://oracle.com/Executive.html">Executive</A></td>
<td>Seattle</td></tr>
<tr><td><A HREF="http://oracle.com/Executive.html">Executive</A></td>
<td>Seattle</td></tr>
<tr><td><A HREF="http://oracle.com/Executive.html">Executive</A></td>
<td>Seattle</td></tr>
<tr><td><A HREF="http://oracle.com/Sales.html">Sales</A></td>
<td>Oxford</td></tr>
<tr><td><A HREF="http://oracle.com/Sales.html">Sales</A></td>
<td>Oxford</td></tr>
<tr><td><A HREF="http://oracle.com/Marketing.html">Marketing</A></td>
<td>Toronto</td></tr>
</table>
<p>

6 rows selected.<br>

SQL&gt; spool off
<br>
</body>
</html>
```

DEPARTMENT_NAME	CITY
Executive	Seattle
Executive	Seattle
Executive	Seattle
Sales	Oxford
Sales	Oxford
Marketing	Toronto

6 rows selected.

The SQLPLUS command in this example contains three layers of nested quotes. From the inside out, these are:

- "2" is a quoted HTML attribute value for BORDER.

- 'BORDER="2"' is the quoted text argument for the TABLE option.

- "HTML ON TABLE 'BORDER="2"'" is the quoted argument for the -MARKUP option.

The nesting of quotes may be different in some operating systems or program scripting languages.

Example 8-1   Creating a Report Interactively

You can create HTML output in an interactive SQL*Plus session using the SET MARKUP command. You can include an embedded style sheet, or any other valid text in the HTML <HEAD> tag. Open a SQL*Plus session and enter the following:

```
SET MARKUP HTML ON SPOOL ON PREFORMAT OFF ENTMAP ON -
HEAD "<TITLE>Department Report</TITLE> -
<STYLE type='text/css'> -
<!-- BODY {background: #FFFFC6} --> -
</STYLE>" -
BODY "TEXT='#FF00Ff'" -
TABLE "WIDTH='90%' BORDER='5'"
```

You use the COLUMN command to control column output. The following COLUMN commands create new heading names for the SQL query output. The first command also turns off entity mapping for the DEPARTMENT_NAME column to allow HTML hyperlinks to be correctly created in this column of the output data:

```
COLUMN DEPARTMENT_NAME HEADING 'DEPARTMENT' ENTMAP OFF
COLUMN CITY HEADING 'CITY'
```

SET MARKUP HTML ON SPOOL ON enables SQL*Plus to write HTML to a spool file. The following SPOOL command triggers the writing of the <HTML> and <BODY> tags to the named file:

```
SPOOL report.html
```

After the SPOOL command, anything entered or displayed on standard output is written to the spool file, report.html.

Enter a SQL query:

```
SELECT '<A HREF="http://oracle.com/'||DEPARTMENT_NAME||'.html">'||
DEPARTMENT_NAME||'</A>' DEPARTMENT_NAME, CITY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
```

Enter the SPOOL OFF command:

```
SPOOL OFF
```

The </BODY> and </HTML> tags are appended to the spool file, report.html, before it is closed.

The output from report.sql is a file, report.html, that can be loaded into a web browser. Open report.html in your web browser. It should appear something like the following:

Example 8-2   Creating a Report using the SQLPLUS Command

Enter the following command at the operating system prompt:

```
SQLPLUS -S -M "HTML ON TABLE 'BORDER="2"'" HR@Ora10g @depart.sql>depart.html
```

where depart.sql contains:

```
SELECT DEPARTMENT_NAME, CITY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
EXIT
```

This example starts SQL*Plus with user "HR", prompts for the HR password, sets HTML ON, sets a BORDER attribute for TABLE, and runs the script depart.sql. The output from depart.sql is a web page which, in this case, has been redirected to the file depart.html using the ">" operating system redirect command (it may be different on your operating system). It could be sent to a web browser if SQL*Plus was called in a web server CGI script. See About Suppressing the Display of SQL*Plus Commands in Reports for information about calling SQL*Plus from a CGI script.

Start your web browser and enter the appropriate URL to open depart.html:

8.1.1.1 HTML Entities

Certain characters, <, >, " and & have a predefined meaning in HTML. In the previous example, you may have noticed that the > character was replaced by > as soon as you entered the SET MARKUP HTML ON command. To enable these characters to be displayed in your web browser, HTML provides character entities to use instead.

Table 8-1   *Equivalent* HTML Entities

Character	HTML Entity	Meaning
<	<	Start HTML tag label
>	>	End HTML tag label
"	"	Double quote
&	&	Ampersand

The web browser displays the > character, but the actual text in the HTML encoded file is the HTML entity, >. The SET MARKUP option, ENTMAP, controls the substitution of HTML entities. ENTMAP is set ON by default. It ensures that the characters <, >, " and & are always replaced by the HTML entities representing these characters. This prevents web browsers from misinterpreting these characters when they occur in your SQL*Plus commands, or in data resulting from your query.

You can set ENTMAP at a global level with SET MARKUP HTML ENTMAP ON, or at a column level with COLUMN *column_name* ENTMAP ON.

8.1.2 Creating CSV Reports

You can enable CSV markup while logging into a user session, by using the -M[ARKUP] CSV ON option at the SQL*Plus command line. For more information, see SQL*Plus Program Syntax. While logged in to a user session, you can enable CSV markup by using the SET MARKUP CSV ON command.

You can specify the delimiter character by using the DELIMITER option. You can also output text without quotes by using QUOTE OFF.

For more information about creating CSV reports, see SET MARKUP CSV ON.

8.1.3 About Suppressing the Display of SQL*Plus Commands in Reports

The SQLPLUS -SILENT option is particularly useful when used in combination with -MARKUP to generate embedded SQL*Plus reports using CGI scripts or operating system scripts. It suppresses the display of SQL*Plus commands and the SQL*Plus banner. The HTML output shows only the data resulting from your SQL query.

You can also use SET ECHO OFF to suppress the display of each command in a script that is executed with the START command.

9

Tuning SQL*Plus

This chapter provides information about how to tune SQL*Plus for better performance. It discusses the following topics:

- About Tracing Statements
- About Collecting Timing Statistics
- Tracing Parallel and Distributed Queries
- Execution Plan Output in Earlier Databases
- About SQL*Plus Script Tuning

For information about tuning Oracle Database, see the *Oracle Database Performance Tuning Guide*.

9.1 About 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.

SQL*Plus report output may differ for DML if dynamic sampling is in effect.

9.1.1 Controlling the Autotrace Report

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

Autotrace Setting	Result
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. If STATISTICS is enabled, query data is still fetched, but not printed.

To use this feature, you must create a PLAN_TABLE table in your schema and then have the PLUSTRACE role granted to you. DBA privileges are required to grant the PLUSTRACE role. For information on how to grant a role and how to create the PLAN_TABLE table, see the *Oracle Database SQL Language Reference*.

For more information about the roles and the PLAN_TABLE, see the *Oracle Database SQL Language Reference* and the AUTOTRACE variable of the SET command.

> **✎ Note:**
>
> SQL*Plus AUTOTRACE does not support switching containers with the
> ALTER SESSION SET CONTAINER option. Statistical data gathered in this
> case may be inconsistent.

Example 9-1   Creating a PLAN_TABLE

Run the following commands from your SQL*Plus session to create the PLAN_TABLE
in the HR schema:

```
CONNECT HR
@$ORACLE_HOME/rdbms/admin/utlxplan.sql

Table created.
```

Example 9-2   Creating the PLUSTRACE Role

Run the following commands from your SQL*Plus session to create the PLUSTRACE
role and grant it to the DBA:

```
CONNECT / AS SYSDBA
@$ORACLE_HOME/sqlplus/admin/plustrce.sql

drop role plustrace;

Role dropped.

create role plustrace;

Role created.

grant plustrace to dba with admin option;

Grant succeeded.
```

Example 9-3   Granting the PLUSTRACE Role

Run the following commands from your SQL*Plus session to grant the PLUSTRACE
role to the HR user:

```
CONNECT / AS SYSDBA
GRANT PLUSTRACE TO HR;

Grant succeeded.
```

9.1.2 Execution Plan

The Execution Plan shows the SQL optimizer's query execution path. Execution Plan
output is generated using EXPLAIN PLAN and DBMS_XPLAN.

9.1.3 Statistics

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

Database Statistic Name	Description
recursive calls	Number of recursive calls generated at both the user and system level. Oracle Database maintains tables used for internal processing. When Oracle Database needs to make a change to these tables, it internally generates an internal SQL statement, which in turn generates a recursive call.
db block gets	Number of times a CURRENT block was requested.
consistent gets	Number of times a consistent read was requested for a block
physical reads	Total number of data blocks read from disk. This number equals the value of "physical reads direct" plus all reads into buffer cache.
redo size	Total amount of redo generated in bytes
bytes sent through Oracle Net Services to client	Total number of bytes sent to the client from the foreground processes.
bytes received through Oracle Net Services from client	Total number of bytes received from the client over Oracle Net.
Oracle Net Services round-trips to/from client	Total number of Oracle Net messages sent to and received from the client
sorts (memory)	Number of sort operations that were performed completely in memory and did not require any disk writes
sorts (disk)	Number of sort operations that required at least one disk write
rows processed	Number of rows processed during the operation

The client referred to in the statistics is SQL*Plus. Oracle Net refers to the generic process communication between SQL*Plus and the server, regardless of whether Oracle Net is installed. You cannot change the default format of the statistics report.

For a more complete list of database statistics, see Statistics Descriptions. For more information about the statistics and how to interpret them, see Gathering Database Statistics.

```
LAST_NAME                    SALARY JOB_TITLE
------------------------ ---------- ---------------------------------
King                          24000 President
De Haan                       17000 Administration Vice President
Kochhar                       17000 Administration Vice President
Partners                      13500 Sales Manager
Russell                       14000 Sales Manager
Hartstein                     13000 Marketing Manager
6 rows selected.


Execution Plan
----------------------------------------------------------
Plan hash value: 2988506077
-----------------------------------------------------------------------------
-
```

```
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)|
Time     |
-----------------------------------------------------------------------
-------
|   0 | SELECT STATEMENT   |          |    6 |   360 |     6  (17)|
00:00:01 |
|*  1 |  HASH JOIN         |          |    6 |   360 |     6  (17)|
00:00:01 |
|*  2 |   TABLE ACCESS FULL| EMPLOYEES|    6 |   204 |     3   (0)|
00:00:01 |
|   3 |   TABLE ACCESS FULL| JOBS     |   19 |   494 |     2   (0)|
00:00:01 |
-----------------------------------------------------------------------
-------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("E"."JOB_ID"="J"."JOB_ID")
   2 - filter("E"."SALARY">12000)

Note
-----
   - dynamic sampling used for this statement

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         10  consistent gets
          0  physical reads
          0  redo size
        706  bytes sent via Oracle Net Services to client
        496  bytes received via Oracle Net Services from client
          2  Oracle Net Services roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          6  rows processed


6 rows selected.

Execution Plan
----------------------------------------------------------
Plan hash value: 2988506077


-----------------------------------------------------------------------
-------
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)|
Time     |
-----------------------------------------------------------------------
-------
|   0 | SELECT STATEMENT   |          |    6 |   360 |     6  (17)|
00:00:01 |
|*  1 |  HASH JOIN         |          |    6 |   360 |     6  (17)|
00:00:01 |
```

```
|*  2 |    TABLE ACCESS FULL| EMPLOYEES|    6 |   204 |     3   (0)| 00:00:01
|
|   3 |    TABLE ACCESS FULL| JOBS     |   19 |   494 |     2   (0)| 00:00:01
|
-------------------------------------------------------------------------------
-

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("E"."JOB_ID"="J"."JOB_ID")
   2 - filter("E"."SALARY">12000)

Note
-----
   - dynamic sampling used for this statement

Statistics
----------------------------------------------------------
           0  recursive calls
           0  db block gets
          10  consistent gets
           0  physical reads
           0  redo size
         706  bytes sent via Oracle Net Services to client
         496  bytes received via Oracle Net Services from client
           2  Oracle Net Services roundtrips to/from client
           0  sorts (memory)
           0  sorts (disk)
           6  rows processed
```

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

> **Note:**
>
> Your output may vary depending on the server version and configuration.

Example 9-4   Tracing Statements for Performance Statistics and Query Execution Path

If the SQL buffer contains the following statement:

```
SELECT E.LAST_NAME, E.SALARY, J.JOB_TITLE
FROM EMPLOYEES E, JOBS J
WHERE E.JOB_ID=J.JOB_ID AND E.SALARY>12000;
```

The statement can be automatically traced when it is run:

```
SET AUTOTRACE ON
/
```

Example 9-5   Tracing Statements Without Displaying Query Data

To trace the same statement without displaying the query data, enter:

```
SET AUTOTRACE TRACEONLY
/
```

9.2 About Collecting Timing Statistics

Use the SQL*Plus TIMING command 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 the TIMING command, and About Tracing Statements for information about using AUTOTRACE to collect statistics.

To delete all timers, enter CLEAR TIMING.

9.3 Tracing Parallel and Distributed Queries

When you trace a statement in a parallel or distributed query, the Execution Plan output depends on the statement you use.

Example 9-6   Tracing Statements With Parallel Query Option

To trace a parallel query running the parallel query option:

```
create table D2_t1 (unique1 number) parallel -
(degree 6);

Table created.

create table D2_t2 (unique1 number) parallel -
(degree 6);

Table created.

create unique index d2_i_unique1 on d2_t1(unique1);

Index created.

set long 500 longchunksize 500
SET AUTOTRACE ON EXPLAIN
SELECT /*+ INDEX(B,D2_I_UNIQUE1) USE_NL(B) ORDERED -
*/ COUNT (A.UNIQUE1)
FROM D2_T2 A, D2_T1 B
WHERE A.UNIQUE1 = B.UNIQUE1;

Execution Plan
---------------------------------------------------------
Plan hash value: 107954098


----------------------------------------------------------------------------
----------------------------
| Id  | Operation              | Name    | Rows |Bytes| Cost(%CPU)| Time
|   TQ  |IN-OUT| PQ Distrib |
----------------------------------------------------------------------------
----------------------------
|   0 | SELECT STATEMENT       |         |    1 |  26 |    1   (0)| 00:00:01
|       |      |            |
|   1 |  SORT AGGREGATE        |         |    1 |  26 |           |
|       |      |            |
```

```
|   2 |   PX COORDINATOR         |         |     |     |     |         |
|     |     |     |         |
|   3 |    PX SEND QC (RANDOM)   | :TQ10001|   1 |  26 |     |         |         |
Q1,01 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE       |         |   1 |  26 |     |         |         |
Q1,01 | PCWP |         |
|   5 |      NESTED LOOPS        |         |   1 |  26 |   1  (0)| 00:00:01 |
Q1,01 | PCWP |         |
|   6 |       PX RECEIVE         |         |     |     |     |         |         |
Q1,01 | PCWP |         |
|   7 |        PX SEND BROADCAST | :TQ10000|     |     |     |         |         |
Q1,00 | P->P | BROADCAST  |
|   8 |         PX BLOCK ITERATOR |        |   1 |  13 |   0  (0)| 00:00:01 |
Q1,00 | PCWC |         |
|   9 |          TABLE ACCESS FULL| D2_T2  |   1 |  13 |   0  (0)| 00:00:01 |
Q1,00 | PCWP |         |
|  10 |         PX BLOCK ITERATOR |        |   1 |  13 |   2  (0)| 00:00:01 |
Q1,01 | PCWC |         |
|* 11 |          TABLE ACCESS FULL | D2_T1 |   1 |  13 |   2  (0)| 00:00:01 |
Q1,01 | PCWP |         |
--------------------------------------------------------------------------------------
----------------------

Predicate Information (identified by operation id):
---------------------------------------------------
  11 - filter("A"."UNIQUE1"="B"."UNIQUE1")

Note
-----
   - dynamic sampling used for this statement
```

Example 9-7   To monitor disk reads and buffer gets.

```
SET AUTOTRACE TRACEONLY STATISTICS
```

The following shows typical results:

```
Statistics
----------------------------------------------------------
        467  recursive calls
         27  db block gets
        147  consistent gets
         20  physical reads
       4548  redo size
        502  bytes sent via Oracle Net Services to client
        496  bytes received via Oracle Net Services from client
          2  Oracle Net Services roundtrips to/from client
         14  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

If consistent gets or physical reads are high relative to the amount of data returned, it
indicates that the query is expensive and needs to be reviewed for optimization. For example,
if you are expecting less than 1,000 rows back and consistent gets is 1,000,000 and physical
reads is 10,000, further optimization is needed.

> **Note:**
>
> You can also monitor disk reads and buffer gets using V$SQL or TKPROF.

9.4 Execution Plan Output in Earlier Databases

Execution Plan output from Oracle Database 9*i* Release 2 (9.2) or earlier is different.

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 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

```
COLUMN PARENT_ID_PLUS_EXP NOPRINT
```

The Execution Plan output is generated using the EXPLAIN PLAN command.

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 Distributed Transactions Concepts 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.

9.5 About SQL*Plus Script Tuning

Most performance benefit comes from tuning SQL queries executed in a script. This is done with tools like SQL*Plus's AUTOTRACE command. It involves restructuring queries to make best use of the Oracle Database SQL optimizer. For information about Tuning SQL statements, see the *Oracle Database Performance Tuning Guide*.

The performance gains made by tuning SQL*Plus-specific commands are smaller, but could be important for some applications. The following system variables and commands can influence SQL*Plus performance.

9.5.1 COLUMN NOPRINT

COLUMN NOPRINT turns off screen output and printing of the column heading and all values selected for the column.

It is better to remove an unneeded column from a SELECT then it is to use COLUMN NOPRINT to stop it displaying. Removing the column from the query means the SQL engine does not need to process it, or need to transfer the column data back to SQL*Plus.

9.5.2 SET APPINFO OFF

Sets automatic registering of scripts through the DBMS_APPLICATION_INFO package. Setting APPINFO OFF prevents administrators monitoring the performance and resource usage of scripts.

If many SQL scripts are being called, then turning APPINFO OFF stops internal SQL*Plus calls to the database DBMS_APPLICATION_INFO package.

9.5.3 SET ARRAYSIZE

Sets the number of rows that SQL*Plus will fetch from the database at one time. Valid values are 1 to 5000.

The effectiveness of setting ARRAYSIZE depends on how well Oracle Database fills network packets and your network latency and throughput. In recent versions of SQL*Plus and Oracle Database, ARRAYSIZE may have little effect. Overlarge sizes can easily take more SQL*Plus memory which may decrease overall performance.

9.5.4 SET DEFINE OFF

SET DEFINE OFF disables the parsing of commands to replace substitution variables with their values.

9.5.5 SET FLUSH OFF

SET FLUSH OFF enables the operating system to buffer output. ON disables buffering and flushes output to the screen. Any benefit from setting FLUSH either ON or OFF depends on your operating system and data. The gain may be marginal.

Use OFF only when you run a script that does not require user interaction and whose output you do not need to see until the script finishes running.

9.5.6 SET LINESIZE

SET LINESIZE sets the total number of characters that SQL*Plus displays on one line before beginning a new line.

Keep LINESIZE as small as possible to avoid extra memory allocations and memory copying.

However, if LINESIZE is too small, columns that cannot fit next to each other are put on separate lines. This may reduce performance significantly.

9.5.7 SET LONGCHUNKSIZE

SET LONGCHUNKSIZE sets the size of the increments SQL*Plus uses to retrieve a BLOB, BFILE, CLOB, LONG, NCLOB or XMLType value.

Experiment with different sizes if LONGS or LOBs are being fetched.

9.5.8 SET PAGESIZE

Sets the number of lines on each page of output.

Increase PAGESIZE to avoid printing headings frequently, or set it to 0 to prevent headings being displayed.

9.5.9 SET SERVEROUTPUT

SET SERVEROUTPUT OFF suppresses the display output (DBMS_OUTPUT.PUT_LINE) of stored procedures or PL/SQL blocks in SQL*Plus.

Setting SERVEROUTPUT OFF stops internal SQL*Plus calls to the DBMS_OUTPUT package done after user SQL statements.

9.5.10 SET SQLPROMPT

Sets the SQL*Plus command prompt.

Use the default prompt, "SQL> " to stop variable substitution occurring each time the prompt is displayed.

9.5.11 SET TAB

Determines how SQL*Plus formats white space in terminal output.

Setting TAB ON causes multiple spaces to be compressed in terminal output. Unless this significantly reduces the written data, the processing required may marginally outweigh any benefit.

9.5.12 SET TERMOUT

SET TERMOUT OFF suppresses the display so that you can spool output from a script without seeing it on the screen.

If both spooling to file and writing to terminal are not required, use SET TERMOUT OFF in SQL scripts to disable terminal output.

9.5.13 SET TRIMOUT ON SET TRIMSPOOL ON

SET TRIMOUT ON or SET TRIMSPOOL ON removes trailing blanks at the end of each displayed or spooled line.

Setting these variables ON can reduce the amount of data written. However, if LINESIZE is optimal, it may be faster to set the variables OFF. The SQL*Plus output line is blank filled throughout the query processing routines so removing the spaces could take extra effort.

9.5.14 UNDEFINE

Deletes substitution variables that you defined either explicitly (with the DEFINE command) or implicitly (with an argument to the START command or COLUMN NEW_VAL|OLD_VAL).

Use the UNDEFINE command to remove unnecessary substitution variables. This can reduce the time taken for any operation that uses '&', new_value or old_value variables.

9.5.15 SET ROWPREFETCH

Minimizes server round trips in a query. The data is prefetched in a result set rows when executing a query. The number of rows to prefetch can be set using this SET ROWPREFETCH This option can reduce round trips by allowing Oracle to transfer query results on return from its internal OCI execute call, removing the need for the subsequent internal OCI fetch call to make another round trip to the DB.

9.5.16 SET STATEMENTCACHE

This option is to cache executed statements in the current session. The benefit of this setting is that it reduces unnecessary parsing time for the same query. Therefore it improves performance when repeatedly executing a query in a session

9.5.17 SET LOBPREFETCH

This option is to improve access of smaller LOBs where LOB data is prefetched and cached. The benefit of this setting is to reduce the number of network round trips to the server, allowing LOB data to be fetched in one round trip when LOB data is within the LOBPREFETCH size defined.

9.5.18 Command Line -FAST Option

This command line option improves performance in general. When this option is used, it changes the following SET options to new values:

- ARRAYSZE 100
- LOBPREFETCH 16384
- PAGESIZE 50000

- • ROWPREFETCH 2
- • STATEMENTCACHE 20

Once logged in, these setting can also be changed manually. Syntax:

```
$ sqlplus -f @emp.sql
```

10
SQL*Plus Security

This chapter describes the available methods for controlling access to database tables, and SQL*Plus commands. It covers the following topics:

- Disabling SQL*Plus_ SQL_ and PL/SQL Commands
- About Creating and Controlling Roles
- About Disabling Commands with SQLPLUS -RESTRICT
- About Program Argument Security

10.1 Disabling SQL*Plus, SQL, and PL/SQL Commands

> **Note:**
>
> Starting with Oracle Database 19c, the SQL*Plus table PRODUCT_USER_PROFILE (PUP table) is desupported. Oracle recommends that you protect data by using Oracle Database settings, so that you ensure consistent security across all client applications.

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:

```
PRODUCT     USERID  ATTRIBUTE  SCOPE  NUMERIC    CHAR      DATE    LONG
                                      VALUE      VALUE     VALUE   VALUE
-------     ------  ---------  -----  --------   ------    -----   -----
SQL*Plus    HR      HOST                         DISABLED
SQL*Plus    %       INSERT                       DISABLED
SQL*Plus    %       UPDATE                       DISABLED
SQL*Plus    %       DELETE                       DISABLED
```

To re-enable commands, delete the row containing the restriction.

SQL*Plus commands that can be disabled:

- ACCEPT
- DEFINE
- PASSWORD
- SHUTDOWN
- APPEND
- DEL
- PAUSE

- SPOOL
- ARCHIVE LOG
- DESCRIBE
- PRINT
- START (@, @@)
- ATTRIBUTE
- DISCONNECT
- PROMPT
- STARTUP
- BREAK
- EDIT
- RECOVER
- STORE
- BTITLE
- EXECUTE
- REMARK
- TIMING
- CHANGE
- EXIT/QUIT
- REPFOOTER
- TTITLE
- CLEAR
- GET
- REPHEADER
- UNDEFINE
- COLUMN
- HELP (?)
- RUN
- VARIABLE
- COMPUTE
- HOST
- SAVE
- WHENEVER OSERROR
- CONNECT
- INPUT
- SET
- WHENEVER SQLERROR

- COPY
- LIST (;)
- SHOW
- XQUERY

SQL commands that can be disabled:

- ALTER
- ANALYZE
- ASSOCIATE
- AUDIT
- CALL
- COMMENT
- COMMIT
- CREATE
- DELETE
- DISASSOCIATE
- DROP
- EXPLAIN
- FLASHBACK
- GRANT
- INSERT
- LOCK
- MERGE
- NOAUDIT
- PURGE
- RENAME
- REVOKE
- ROLLBACK
- SAVEPOINT
- SELECT
- SET CONSTRAINTS
- SET ROLE
- SET TRANSACTION
- TRUNCATE
- UPDATE
- VALIDATE

You can disable the following PL/SQL commands:

- BEGIN

- DECLARE

> **Note:**
>
> - Disabling HOST disables the operating system alias for HOST, such as $ on Windows, and ! on UNIX.
> - Disabling LIST disables ; and numbers (numbers entered to go to that line in a script).
> - You must disable HELP and ? separately to disable access to command-line help.
> - Disabling the SQL*Plus SET command also disables SQL SET CONSTRAINTS, SET ROLE and SET TRANSACTION.
> - Disabling SQL*Plus START also disables @ and @@.
> - Disabling BEGIN and DECLARE does not prevent the use of SQL*Plus EXECUTE to run PL/SQL. EXECUTE must be disabled separately.
> - Disabling EXIT/QUIT is not recommended. If disabled, terminate a command-line session by sending an EOF character such as Ctrl+D in UNIX or Ctrl+Z in Windows. Otherwise, terminate a session by terminating the SQL*Plus process. If disabled, the EXIT operation in WHENEVER OSERROR and WHENEVER SQLERROR is also disabled.

1. Log in as SYSTEM with the command

```
SQLPLUS SYSTEM
```

2. Insert a row into the PUP table with the command:

```
INSERT INTO PRODUCT_USER_PROFILE
VALUES ('SQL*Plus', 'HR', 'SELECT', NULL, NULL, 'DISABLED', NULL, NULL);
```

3. Connect as HR and try to SELECT something:

```
CONNECT HR
SELECT * FROM EMP_DETAILS_VIEW;
```

This command causes the following error message:

```
SP2-0544: Command SELECT disabled in Product User Profile
```

4. To delete this row and remove the restriction from the user HR, CONNECT again as SYSTEM and enter:

```
DELETE FROM PRODUCT_USER_PROFILE WHERE USERID = 'HR';
```

10.2 About Creating and Controlling Roles

You can use SQL commands to create and control access to roles to provide security for your database tables. 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.

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.

10.2.1 About 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 through 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 the PUP table to disable the SET ROLE command. You also need to disable the BEGIN and SQL*Plus EXECUTE commands to prevent application users setting application roles through a PL/SQL block. This gives 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 *Oracle Database SQL Language Reference* and *Oracle Database Administrator's Guide*.

10.2.2 About Disabling User Roles

To disable a role for a given user, insert a row in the PUP table 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:

```
PRODUCT     USERID  ATTRIBUTE  SCOPE   NUMERIC     CHAR      DATE     LONG
                                       VALUE       VALUE     VALUE    VALUE

-------     ------  ---------  -----   --------    ------    -----    -----
SQL*Plus    HR      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.

To re-enable roles, delete the row containing the restriction.

See About Disabling SET ROLE for more information.

Chapter 10
About Disabling Commands with SQLPLUS -RESTRICT

10.3 About Disabling Commands with SQLPLUS -RESTRICT

The RESTRICT option enables you to disable certain commands that interact with the operating system. However, commands disabled with the -RESTRICT option are disabled even when no connection to a server exists, and remain disabled until SQL*Plus terminates.

The following table shows which commands are disabled in each restriction level.

Command	Level 1	Level 2	Level 3
EDIT	disabled	disabled	disabled
GET			disabled
HOST	disabled	disabled	disabled
SAVE		disabled	disabled
SPOOL		disabled	disabled
START			disabled
STORE		disabled	disabled

> **Note:**
>
> - Disabling HOST also disables your operating system's alias for HOST, such as $ on Windows, and ! on UNIX.
> - Disabling the SQL*Plus START command will also disable the SQL*Plus @ and @@ commands.

For more information about the RESTRICT option, see the SQLPLUS RESTRICT Option.

10.4 About Program Argument Security

Some operating systems allow any user to see what programs are being run. If the display also shows command-line arguments, it may be possible to view the usernames and passwords of other SQL*Plus users.

For example, on many UNIX or Linux systems the ps command shows program arguments. To stop passwords being displayed depends on how you use SQL*Plus.

- To run SQL*Plus interactively, always wait for SQL*Plus to prompt for connection information, particularly your password.
- To run a batch SQL script from a UNIX shell script, set environment variables MYUSERNAME and MYPASSWORD to the appropriate values. Run a shell script containing:

```
sqlplus /nolog <<EOF
connect $MYUSERNAME/$MYPASSWORD
```

```
select ...
EOF
```

- To run a batch SQL script, hard code the username and password as the first line of the SQL script. Then call the script with:

```
sqlplus @myscript.sql
```

When SQL*Plus is started like this, it uses the first line of the script as the username/password@connection_identifier string.

Avoid storing your username and password in files or scripts. If you do store your username and password in a file or script, ensure that the file or script is secured from non-authorized access.

11

Database Administration with SQL*Plus

This chapter provides a brief overview of the database administration tools available in SQL*Plus, and discusses the following topics:

- Overview
- Introduction to Database Startup and Shutdown
- Redo Log Files
- Database Recovery

This chapter is intended for use by database administrators. You must have database administrator privileges to use this functionality.

For more information on database administration, see the *Oracle Database Concepts* manual.

11.1 Overview

Special operations such as starting up or shutting down a database are performed by a database administrator (DBA). The DBA has certain privileges that are not assigned to normal users. The commands outlined in this chapter would normally be used by a DBA.

For more information about security and roles in SQL*Plus, see SQL*Plus Security.

11.2 Introduction to Database Startup and Shutdown

An Oracle database may not always be available to all users. To open or close a database, or to start up or shut down an instance, you must have DBA privileges or be connected as SYSOPER or SYSDBA. Other users cannot change the current status of an Oracle database.

11.2.1 Database Startup

To start a database:

1. Start an instance

   An instance controls the background processes and the allocation of memory area to access an Oracle database.

2. Mount the database

   Mounting the database associates it with a previously started instance.

3. Open the database

   Opening the database makes it available for normal database operations.

Example 11-1   Starting an Instance

To start an Oracle Database instance, without mounting the database, enter

```
STARTUP NOMOUNT
```

Example 11-2   Mounting the Database

To start an instance, mount the database, but leave the database closed, enter

```
STARTUP MOUNT
```

Example 11-3   Opening the Database

To start an instance using the Oracle Database Server parameter file INITSALES.ORA, mount and open the database named SALES, and restrict access to database administrators, enter

```
STARTUP OPEN sales PFILE=INITSALES.ORA RESTRICT
```

where SALES is the database name specified in the DB_NAME parameter in the INITSALES.ORA parameter file.

For more information about database startup, see Starting Up and Shutting Down. For more information about starting a database, see the STARTUP command.

11.2.2 PDB Startup

A Pluggable Database (PDB) is a self-contained collection of schemas and schema objects that exist inside a Consolidated Database.

To start a pluggable database:

1. Start SQL*Plus with the `/NOLOG` argument:

    ```
    sqlplus /nolog
    ```

2. Issue a `CONNECT` command using easy connect or a net service name to connect to the PDB.

3. Issue a STARTUP command.

Another way to open a pluggable database is to connect to the CDB and use the following command:

```
ALTER PLUGGABLE DATABASE pdbname OPEN;
```

For more information about database startup, see Starting Up and Shutting Down. For more information about starting a database, see the STARTUP command.

11.2.3 Database Shutdown

Shutting down a database involves three steps:

1. Closing the database

    When a database is closed, all database and recovery data in the SGA are written to the datafiles and redo log files, and all online datafiles are closed.

2. Dismounting the database

    Dismounting the database disassociates the database from an instance and closes the control files of the database.

3. Shutting down the instance

Shutting down an instance reclaims the SGA from memory and terminates the background Oracle Database processes that constitute an Oracle Database instance.

Example 11-4   Shutting Down the Database

To shut down the database normally after it has been opened and mounted, enter

```
SHUTDOWN
```

For more information about database shutdown, see Shutting Down a Database. For information about stopping a database, see the SHUTDOWN command.

```
Database closed.
Database dismounted.
ORACLE instance shut down.
```

11.2.4 PDB Shutdown

To shutdown a pluggable database (PDB):

1. Connect to the PDB with the required privileges.

2. Run the `SHUTDOWN` command.

> ✎ **Note:**
>
> When the current container is a PDB, the `SHUTDOWN` command only closes the PDB, not the CDB instance.

For more information about database startup, see the *Oracle Database Administrator's Guide* guide. For more information about starting a database, see the STARTUP command.

11.3 Redo Log Files

Every Oracle database has a set of two or more redo log files. The set of redo log files for a database is collectively referred to as the database's redo log.

The redo log is used to record changes made to data. If, for example, there is a database failure, the redo log is used to recover the database. To protect against a failure involving the redo log itself, Oracle Database has a mirrored redo log so that two or more copies of the redo log can be maintained on different disks.

11.3.1 ARCHIVELOG Mode

Operating a database in ARCHIVELOG mode enables the archiving of the online redo log.

The SQL ALTER SYSTEM command enables a complete recovery from disk failure as well as instance failure, because all changes made to the database are permanently saved in an archived redo log.

For more information about redo log files and database archiving modes, see the ARCHIVE LOG command.

To list the details of the current log file being archived, enter

```
ARCHIVE LOG LIST
```

```
Database log mode             Archive Mode
Automatic archival            Enabled
Archive destination           /vobs/oracle/dbs/arch
Oldest online log sequence    221
Next log sequence to archive  222
Current log sequence          222
```

11.4 Database Recovery

If a damaged database is in ARCHIVELOG mode, it is a candidate for either complete media recovery or incomplete media recovery operations. To begin media recovery operations use the RECOVER command. For more information about recovering data, see the RECOVER command.

In order to begin recovery operations, you must have DBA privileges.

To recover the database up to a specified time using a control backup file, enter

```
RECOVER DATABASE UNTIL TIME '1998-11-23:12:47:30'-
USING BACKUP CONTROLFILE
```

To recover two offline tablespaces, enter

```
RECOVER TABLESPACE ts1, ts2
```

Make sure that the tablespaces you are interested in recovering have been taken offline, before proceeding with recovery for those tablespaces.

12

SQL*Plus Globalization Support

Globalization support enables the storing, processing and retrieval of data in native languages. The languages that can be stored in an Oracle database are encoded by Oracle Database-supported character sets. Globalization support ensures that database utilities, error messages, sort order, and date, time, monetary, numeric, and calendar conventions adjust to the native language and locale.

Topics:

- About Configuring Globalization Support in Command-line SQL*Plus
- NLS_LANG Environment Variable

For more information on globalization support, see the Oracle Technology Network globalization notes at `http://www.oracle.com/technetwork/products/globalization/`

and see Overview of Globalization Support.

12.1 About Configuring Globalization Support in Command-line SQL*Plus

SQL*Plus supports multiple languages through the NLS_LANG environment variable. To display another language in SQL*Plus, before starting SQL*Plus you must configure:

- NLS_LANG in the SQL*Plus client environment.
- The Oracle Database during installation.

12.1.1 SQL*Plus Client

The SQL*Plus client environment is configured by setting the NLS_LANG environment variable which is read by SQL*Plus at startup.

12.1.2 Oracle Database

The Oracle Database environment is configured by creating the database with the required character set.

12.2 NLS_LANG Environment Variable

The NLS_LANG environment variable has three components, each controlling a subset of the globalization features.

Your operating system and keyboard must be able to support the character set you have chosen. You may need to install additional support software. For more information about NLS_LANG, and software support, see Setting Up a Globalization Support Environment.

Setting up locale specific behavior on the SQL*Plus client is achieved with the use of NLS parameters. These parameters may be specified in a number of ways, including as an initialization parameter on the server. For settings that control the behavior of the server, see NLS Database Parameters.

NLS_LANG has the syntax:

```
NLS_LANG = language_territory.charset
```

where *language* specifies the conventions to use for Oracle Database messages, sorting order, day and month names. For example, to receive messages in Japanese, set *language* to JAPANESE. If *language* is not set, it defaults to AMERICAN.

where *territory* specifies the convention for default dates, and for monetary, and numeric formats. For example to use the Japanese territory format, set *territory* to JAPAN. If *territory* is not set, the default value is derived from the language value, and so is set to AMERICA.

where, in SQL*Plus command-line, *charset* specifies the character set encoding used by SQL*Plus for data processing, and is generally suited to that of the users terminal. Illogical combinations can be set, but will not work. For example, Japanese cannot be supported using a Western European character set such as:

```
NLS_LANG=JAPANESE_JAPAN.WE8DEC
```

However, Japanese could be supported with the Unicode character set. For example:

```
NLS_LANG=JAPANESE_JAPAN.UTF8
```

12.2.1 Viewing NLS_LANG Settings

You can view the NLS_LANG setting by entering the SELECT command:

```
SELECT * FROM NLS_SESSION_PARAMETERS;
```

The NLS_TERRITORY and NLS_LANGUAGE values correspond to the language and territory components of the NLS_LANG variable.

You can also obtain a list of valid values for the NLS_SORT, NLS_LANGUAGE, NLS_TERRITORY and NLS_CHARACTERSET by querying the NLS dynamic performance view table V$NLS_VALID_VALUES.

12.3 Setting NLS_LANG

You can set the NLS_LANG environment variable to control globalization features.

Example 12-1   Configuring Japanese Support in SQL*Plus on Windows

1. Ensure you have exited your current SQL*Plus session.

2. Open System from Start > Settings > Control Panel.

3. Click the Advanced tab and select Environment Variables.

4. Create a new environment variable, NLS_LANG, with a value of Japanese_Japan.JA16SJIS.

5. You may need to restart Windows for this setting to take effect.

Example 12-2   Configuring Japanese Support in SQL*Plus on UNIX

1. Ensure you have exited your current SQL*Plus session.

2. Set the NLS_LANG variable using either set or setenv depending on the UNIX shell you are using. For example, in csh, you would enter:

```
setenv NLS_LANG Japanese_Japan.UTF8
```

or

```
setenv NLS_LANG Japanese_Japan.JA16SJIS
```

or

```
setenv NLS_LANG Japanese_Japan.JA16EUC
```

The locale setting of your UNIX terminal determines the exact value of the NLS_LANG parameter. For more information on the NLS_LANG setting, see Specifying the Value of NLS_LANG.

Example 12-3   Configuring Japanese Support in Oracle Database

To store data in the Japanese character set using UTF-8 character encoding, ensure that the Oracle database has been created with the AL32UTF8 character set. See your Oracle Database Installation Guide for information about creating your database in a character set other than US7ASCII.

Part III
SQL*Plus Reference

Part III contains the SQL*Plus command reference.

SQL*Plus Command Reference

ORACLE®

13

SQL*Plus Command Reference

This chapter contains descriptions of the SQL*Plus commands listed alphabetically. Each description contains the following parts:

Section	Description
Syntax	Shows how to enter the command and provides a brief description of the basic uses of the command.
Terms	Describes the function of each term or clause appearing in the syntax.
Usage	Provides additional information on uses of the command and on how the command works.
Examples	Gives one or more examples of the command.

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 press Return. If you wish, however, you can enter a semicolon at the end of a SQL*Plus command.

13.1 SQL*Plus Command Summary

Command	Description
@ (at sign)	Runs SQL*Plus statements in the specified script. The script can be called from the local file system or from a web server.
@@ (double at sign)	Runs a script. This command is similar to the @ (at sign) command It is useful for running nested scripts because it looks for the specified script in the same path as the calling script.
/ (slash)	Executes the SQL command or PL/SQL block.
ACCEPT	Reads a line of input and stores it in a given substitution variable.
APPEND	Adds specified text to the end of the current line in the buffer.
ARCHIVE LOG	Displays information about redo log files.
ARGUMENT	Customizes the input prompt and default values when parameter(s) are not passed to the script. You can now control what the input prompt should be and when to use the default value for the parameters.
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.

Command	Description
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 clause 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.
CONFIG	Generates the config store JSON file from the `tnsnames.ora` file.
CONNECT	Connects a given user to Oracle Database.
COPY	Copies results from a query to a table in the same or another database.
DEFINE	Specifies a substitution variable and assigns it a CHAR value, or lists the value and variable type of a single variable or all variables.
DEL	Deletes one more lines of the buffer.
DESCRIBE	Lists the column definitions for the specified table, view, or synonym or the specifications for the specified function procedure.
DISCONNECT	Commits pending changes to the database and logs the current user off Oracle Database, but does not exit SQL*Plus.
EDIT	Invokes an 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 an operating system file into the buffer.
HELP	Accesses the SQL*Plus command-line help system.
HISTORY	Recalls the history of commands, SQL*Plus commands, and SQL or PL/SQL statements issued in the current SQL*Plus session.
HOST	Executes an operating system command without leaving SQL*Plus.
INPUT	Adds one or more new lines after the current line in the buffer.

Command	Description
LIST	Lists one or more lines of the buffer.
OERR	Displays the cause and action details for the provided facility name and the error number.
PASSWORD	Enables a password to be changed without echoing the password on an input device.
PAUSE	Displays the specified text, then waits for the user to press Return.
PING	Pings the database or database network listener to check availability.
PRINT	Displays the current value of a bind variable.
PROMPT	Sends the specified message to the user's screen.
EXIT	Terminates SQL*Plus and returns control to the operating system QUIT is identical to EXIT.
RECOVER	Performs media recovery on one or more tablespaces, one or more datafiles, or the entire database.
REMARK	Begins a comment in a script.
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 runs the SQL command or PL/SQL block currently stored in the SQL buffer.
SAVE	Saves the contents of the buffer in an operating system file (a script).
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.
SHOW CONNECTION	Lists the Oracle Net Service Names present in the `tnsnames.ora` file and resolves the given net service name to the connection string.
SHUTDOWN	Shuts down a currently running Oracle Database instance.
SPOOL	Stores query results in an operating system file and, optionally, sends the file to a printer.
START	Runs the SQL statements in the specified script. The script can be called from a local file system or a web server in SQL*Plus command-line.
STARTUP	Starts an Oracle Database instance and optionally mounts and opens a database.

Command	Description
STORE	Saves attributes of the current SQL*Plus environment in an operating system script.
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 substitution 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, or lists the current display characteristics for a single variable or all variables.
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.
XQUERY	Runs an XQuery 1.0 statement.

13.2 @ (at sign)

Syntax

```
@{url | file_name[.ext] } [arg...]
```

Runs the SQL*Plus statements in the specified script. The script can be called from the local file system or from a web server.

The @ command functions similarly to @@ and START.

Terms

url

Specifies the Uniform Resource Locator of a script to run on the specified web server. SQL*Plus supports HTTP and FTP protocols, but not HTTPS. HTTP authentication in the form http://*username:password@machine_name.domain*... is not supported in this release.

file_name[.ext]

Represents the script 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 SET SUF[FIX] {SQL | text}.

When you enter *@file_name.ext*, SQL*Plus searches for a file with that filename and extension in the current default directory. If SQL*Plus does not find the file in the current directory, it searches a system-dependent path to find it. Some operating systems may not support the path search. See the platform-specific Oracle documentation 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 script. If you enter one or more arguments, SQL*Plus substitutes the values into the parameters (&1, &2, and so forth) in the script. 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 given by the arguments; if you run the script again in this session, you can enter new arguments or omit the arguments to use the current values. For more information on using parameters, see Using Substitution Variables.

Usage

All previous settings like COLUMN command settings stay in effect when the script starts. If the script changes any setting, this new value stays in effect after the script has finished.

You can include in a script any command you would normally enter interactively (typically, SQL, SQL*Plus commands, or PL/SQL blocks).

If the START command is disabled (see Disabling SQL*Plus_ SQL_ and PL/SQL Commands), this will also disable the @ command. See START for information on the START command.

SQL*Plus removes the SQLTERMINATOR (a semicolon by default) before the @ command is issued. If you require a semicolon in your command, add a second SQLTERMINATOR. See SET SQLT[ERMINATOR] {; | c | ON | OFF} for more information.

Examples

To run a script named PRINTRPT with the extension SQL, enter

```
@PRINTRPT
```

To run a script named WKRPT with the extension QRY, enter

```
@WKRPT.QRY
```

You can run a script named YEAREND specified by a URL, and pass values to variables referenced in YEAREND in the usual way:

```
@HTTP://machine_name.domain:port/YEAREND.SQL VAL1 VAL2
@FTP://machine_name.domain:port/YEAREND.SQL  VAL1 VAL2
```

On a web server configured to serve SQL reports, you could request SQL*Plus to execute a dynamic script with:

```
@HTTP://machine_name.domain:port/SCRIPTSERVER?ENDOFYEAR VAL1 VAL2
```

13.3 @@ (double at sign)

Syntax

```
@@{url | file_name[.ext] } [arg...]
```

Runs a script. This command is almost identical to the @ (at sign) command. When running nested scripts it looks for nested scripts in the same path or *url* as the calling script. The @@ command functions similarly to @ and START.

Terms

url

Specifies the Uniform Resource Locator of a script to run on the specified web server. SQL*Plus supports HTTP and FTP protocols, but not HTTPS. HTTP authentication in the form http://*username:password@machine_name.domain*... is not supported in this release.

file_name[.*ext*]

Represents the nested script 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 SET SUF[FIX] {SQL | text}.

When you enter *@@file_name.ext* from within a script, SQL*Plus runs *file_name.ext* from the same directory as the script.

When you enter *@@file_name.ext* interactively, SQL*Plus runs *file_name.ext* from the current working directory or from the same *url* as the script from which it was called. If SQL*Plus does not find the file, it searches a system-dependent path to find the file. Some operating systems may not support the path search. See the platform-specific Oracle documentation 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 script. If you enter one or more arguments, SQL*Plus substitutes the values into the parameters (&1, &2, and so forth) in the script. 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 given by the arguments. If you run the script again in this session, you can enter new arguments or omit the arguments to use the current values. For more information on using parameters, see Using Substitution Variables.

Usage

All previous settings like COLUMN command settings stay in effect when the script starts. If the script changes any setting, the new value stays in effect after the script has finished.

You can include in a script any command you would normally enter interactively (typically, SQL or SQL*Plus commands).

If the START command is disabled (see Disabling SQL*Plus_ SQL_ and PL/SQL Commands), this will also disable the @@ command. For more information, see the SPOOL 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 SET SQLT[ERMINATOR] {; | c | ON | OFF} for more information.

Examples

Suppose that you have the following script named PRINTRPT:

```
SELECT DEPARTMENT_ID, CITY FROM EMP_DETAILS_VIEW WHERE SALARY>12000;
@EMPRPT.SQL
@@ WKRPT.SQL
```

When you START PRINTRPT and it reaches the @ command, it looks for the script named EMPRPT in the current working directory and runs it. When PRINTRPT reaches the @@ command, it looks for the script named WKRPT in the same path as PRINTRPT and runs it.

Suppose that the same script PRINTRPT was located on a web server and you ran it with START HTTP://*machine_name.domain:port*/PRINTRPT. When it reaches the @ command, it looks for the script named EMPRPT in the current working directory and runs it. When PRINTRPT reaches the @@ command, it looks for the script named WKRPT in the same *url* as PRINTRPT, HTTP://*machine_name.domain:port*/WKRPT.SQL and runs it.

13.4 / (slash)

Syntax

/(slash)

Executes the most recently executed SQL command or PL/SQL block which is stored in the SQL buffer.

Usage

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.

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.

Examples

Type the following SQL script:

```
SELECT CITY, COUNTRY_NAMEFROM EMP_DETAILS_VIEWWHERE SALARY=12000;
```

Enter a slash (/) to re-execute the command in the buffer:

```
/
```

```
CITY                          COUNTRY_NAME
----------------------------- -------------------------------------
Seattle                       United States of America
Oxford                        United Kingdom
Seattle                       United States of America
```

13.5 ACCEPT

Syntax

```
ACC[EPT] variable [NUM[BER] | CHAR | DATE | BINARY_FLOAT | BINARY_DOUBLE]
[FOR[MAT] format] [DEF[AULT] default] [PROMPT text|NOPR[OMPT]] [HIDE]
```

Reads a line of input and stores it in a given substitution variable.

Terms

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 *variable* a NUMBER datatype. If the reply does not match the datatype, ACCEPT gives an error message and prompts again.

`CHAR`

Makes the *variable* a CHAR datatype. The maximum CHAR length 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.

`BINARY_FLOAT`

Makes the *variable* a BINARY_FLOAT datatype. BINARY_FLOAT is a floating-point number that conforms substantially with the Institute for Electrical and Electronics Engineers (IEEE) Standard for Binary Floating-Point Arithmetic, IEEE Standard 754-1985.

`BINARY_DOUBLE`

Makes the *variable* a BINARY_DOUBLE datatype. BINARY_DOUBLE is a floating-point number that conforms substantially with the Institute for Electrical and Electronics Engineers (IEEE) Standard for Binary Floating-Point Arithmetic, IEEE Standard 754-1985.

`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. If an attempt is made to enter more characters than are specified by the char format, an error message is given and the value must be reentered. If an attempt is made to enter a greater number precision than is specified by the number format, an error message is given and the value must be reentered. The format element must be a text constant such as A10 or 9.999. See COLUMN FORMAT for a complete list of format elements.

Oracle Database date formats such as "dd/mm/yy" are valid when the datatype is DATE. DATE without a specified format defaults to the NLS_DATE_FORMAT of the current session. SeeFormat Models for information on Oracle Database 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 for more information.

Examples

To display the prompt "Password: ", place the reply in a CHAR variable named PSWD, and suppress the display, enter

```
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

```
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, HIRED, with the format "dd/mm/yyyy" and a default of "01/01/2003", enter

```
ACCEPT hired DATE FORMAT 'dd/mm/yyyy' DEFAULT '01/01/2003'–
PROMPT 'Enter date hired:  '
```

To display the prompt "Enter employee lastname: " and place the reply in a CHAR variable named LASTNAME, enter

```
ACCEPT lastname CHAR FORMAT 'A20' –
PROMPT 'Enter employee lastname:  '
```

13.6 APPEND

Syntax

```
A[PPEND] text
```

where *text* represents the text to append.

Adds specified text to the end of the current line in the SQL buffer.

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 comma delimiter, a space and the column name CITY to the first line of the buffer, make that line the current line by listing the line as follows:

```
1

1* SELECT DEPARTMENT_ID
```

Now enter APPEND:

```
APPEND , CITY
1

1* SELECT DEPARTMENT_ID, CITY
```

To append a semicolon to the line, enter

```
APPEND ;;
```

SQL*Plus appends the first semicolon to the line and interprets the second as the terminator for the APPEND command.

13.7 ARCHIVE LOG

Syntax

```
ARCHIVE LOG LIST
```

Displays information about redo log files.

Terms

```
LIST
```

Requests a display that shows the range of redo log files to be archived, the current log file group's sequence number, and the current archive destination (specified by either the optional command text or by the initialization parameter LOG_ARCHIVE_DEST).

If you are using both ARCHIVELOG mode and automatic archiving, the display might appear like:

```
ARCHIVE LOG LIST

Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            /vobs/oracle/dbs/arch
Oldest online log sequence     221
Next log sequence to archive   222
Current log sequence           222
```

Since the log sequence number of the current log group and the next log group to archive are the same, automatic archival has archived all log groups up to the current one.

If you are using ARCHIVELOG but have disabled automatic archiving, the last three lines might look like:

```
Oldest online log sequence          222
Next log sequence to archive         222
Current log sequence                 225
```

If you are using NOARCHIVELOG mode, the "next log sequence to archive" line is suppressed.

The log sequence increments every time the Log Writer begins to write to another redo log file group; it does not indicate the number of logs being used. Every time an online redo log file group is reused, the contents are assigned a new log sequence number.

Usage

You must be connected to an open Oracle database as SYSOPER, or SYSDBA. For information about connecting to the database, see the CONNECT command.

For information about specifying archive destinations, see your platform-specific Oracle Database documentation.

> **Note:**
>
> ARCHIVE LOG LIST only applies to the current instance. To START and STOP archiving, use the SQL command ALTER SYSTEM. For more information about using SQL commands, see the *Oracle Database SQL Language Reference*.

13.8 ARGUMENT

Syntax

```
ARGUMENT argument_number [PROMPT text] [DEFAULT text] [HIDE]
```

Customizes the default input prompt text and assigns default values to the parameters when arguments are not passed while executing the script.

Terms

```
argument_number
```

Refers to the argument position.

```
PROMPT
```

Refers to the customized text for the user input.

```
DEFAULT
```

Set the default value for the parameter when input value is not provided when prompted or not pass in as an argument to the script.

```
HIDE
```

Masks the input value. It applies to the `PROMPT` option.

Usage Notes

A prompt is displayed when a parameter has not been defined using any of the following:

- Passed as a script argument

- Set using the `DEFINE` command

- Set using the `COLUMN_OLD` and the `NEW_VALUE` variables

When executing a script, the `PROMPT` and the `DEFAULT` options work as follows:

- When the `PROMPT` text is specified, the text is displayed instead of the default prompt text.

- When the `PROMPT` text is specified with the `HIDE` option, you are prompted for input, but the input value is not displayed.

- When both the `PROMPT` and the `DEFAULT` options are set but the user does not provide a value when prompted, the default value is used. For example, if the user presses the **Enter** key without specifying a value.

- When the `DEFAULT` option is set and the `PROMPT` option is not set, the default value is used. In this case, the user is not prompted for input.

- When the `PROMPT` option is set and the argument value is not passed to the script, a customized input prompt text is displayed instead of the default prompt text.

- When an argument is passed to the script, its value is used. In this case, SQL*Plus neither prompts for input nor uses the default value.

Examples

If the script test1.sql contains:

```
ARGUMENT 1 PROMPT "Enter value for Arg1:"
DEFINE arg1 = '&1';
SELECT ENAME FROM EMP WHERE EMPNO LIKE '&arg1';
```

When you execute this script without providing an argument, it will prompt for input. You can then enter a value, as shown in the following example:

```
@test1.sql
Enter value for Arg1: 7499
```

The script displays the following output:

```
ENAME
------
ALLEN
```

If the script test2.sql contains:

```
ARGUMENT 1 PROMPT "Enter value for Arg1: " default "%"
DEFINE arg1='&1';
SELECT ENAME FROM EMP WHERE EMPNO LIKE '&arg1';
```

When you execute this script without providing an argument, it will prompt for input. Press the **ENTER** key without entering any input value:

```
@test2.sql
Enter value for Arg1:
```

The default input value (%) is used. The script displays the following output:

```
ENAME
----------
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER

14 rows selected.
```

If the script test3.sql contains:

```
ARGUMENT 1 PROMPT "Enter value for Arg1: "
DEFINE arg1='&1';
SELECT ENAME FROM EMP WHERE EMPNO LIKE '&arg1';
```

When you execute this script without providing an argument, it will prompt for input. Press the **ENTER** key without entering any input value:

```
@test3.sql
Enter value for Arg1:
```

The script displays the following output:

```
No rows selected.
```

If the script test4.sql contains:

```
ARGUMENT 1 DEFAULT "7499"
DEFINE arg1 = '&1'
SELECT ENAME FROM EMP WHERE EMPNO LIKE'&arg1';
```

When you execute this script without providing an argument, the default value is used:

```
@test4.sql
```

The script displays the following output:

```
ENAME
----------
ALLEN
```

When you execute this script with an argument, the argument value is used:

```
@test4.sql 7521
```

The script displays the following output:

```
ENAME
----------
WARD
```

If the script test5.sql contains:

```
ARGUMENT 1 PROMPT "enter value for Arg1: " default '%' hide
DEFINE arg1 = '&1';
SELECT ENAME FROM EMP WHERE EMPNO LIKE '&1';
```

When you execute this script without providing an argument, it will prompt for input. Since the HIDE option is specified, the value that you entered is masked:

```
@test5.sql
Enter value for Arg1: ****
```

The script displays the following output:

```
ENAME
----------
WARD
```

13.9 ATTRIBUTE

Syntax

```
ATTR[IBUTE] [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
```

Specifies display characteristics for a given attribute of an Object Type column, such as the format of NUMBER data. Columns and attributes should not have the same names as they share a common namespace.

Also lists the current display characteristics for a single attribute or all attributes.

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.

Terms

```
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* applies 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.

```
ON | 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

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 LAST_NAME attribute of the Object Type EMPLOYEE_TYPE twenty characters wide, enter

```
ATTRIBUTE EMPLOYEE_TYPE.LAST_NAME FORMAT A20
```

To format the SALARY attribute of the Object Type EMPLOYEE_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

```
ATTRIBUTE EMPLOYEE_TYPE.SALARY FORMAT $9,999,990.99
```

13.10 BREAK

Syntax

```
BRE[AK] [ON report_element [action [action]]] ...
```

where *report_element* has the syntax {*column|expr*|ROW|REPORT}

and *action* has the syntax [SKI[P] *n*|[SKI[P]] PAGE] [NODUP[LICATES]| DUP[LICATES]]

Specifies where changes occur in a report and the formatting action to perform, 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 the COMPUTE command.

Enter BREAK with no clauses to list the current BREAK definition.

Terms

```
ON column [action [action]]
```

When you include actions, specifies actions 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, or the end of a report

When you omit actions, 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

```
BREAK ON DEPARTMENT_ID SKIP PAGE ON JOB_ID -
SKIP 1 ON SALARY SKIP 1
```

the first ON clause represents the outermost break (in this case, ON DEPARTMENT_ID) and the last ON clause represents the innermost break (in this case, ON SALARY). SQL*Plus searches each row of output for the specified breaks, 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 DEPARTMENT_ID, then JOB_ID, then SALARY.

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 SALARY toward SKIP PAGE for ON DEPARTMENT_ID). SQL*Plus executes each action up to and including the action specified for the first break encountered in the initial search.

If, for example, in a given row the value of JOB_ID changes—but the values of DEPARTMENT_ID and SALARY remain the same—SQL*Plus skips two lines before printing the row (one as a result of SKIP 1 ON SALARY and one as a result of SKIP 1 ON JOB_ID).

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.

If the BREAK command specified earlier in this section is used, the following SELECT command produces meaningful results:

```
SELECT DEPARTMENT_ID, JOB_ID, SALARY, LAST_NAME
FROM EMP_DETAILS_VIEW
WHERE SALARY > 12000
ORDER BY DEPARTMENT_ID, JOB_ID, SALARY, LAST_NAME;
```

All rows with the same DEPARTMENT_ID print together on one page, and within that page all rows with the same JOB_ID print in groups. Within each group of jobs, those jobs with the same SALARY print in groups. Breaks in LAST_NAME cause no action because LAST_NAME does not appear in the BREAK command.

In BREAK commands, nulls are considered equal to each other, but not equal to anything else. This is different to the treatment of nulls in WHERE clauses.

```
ON expr [action [action]]
```

When you include actions, specifies actions for SQL*Plus to take when the value of the expression changes.

When you omit actions, BREAK ON *expr* suppresses printing of duplicate values of *expr* and marks 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 for ON *column* also applies to ON *expr*.

```
ON ROW [action [action]]
```

When you include actions, specifies actions 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.

```
SKI[P] n
```

Skips *n* lines before printing the row where the break occurred. BREAK SKIP *n* does not work in SET MARKUP HTML ON mode unless PREFORMAT is SET ON.

```
[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 with 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

Each new BREAK command you enter replaces the preceding one.

To remove the BREAK command, use CLEAR BREAKS.

Examples

To produce a report that prints duplicate job values, prints the average of SALARY, and additionally prints the sum of SALARY, you could enter the following commands. (The example selects departments 50 and 80 and the jobs of clerk and salesman only.)

```
BREAK ON DEPARTMENT_ID ON JOB_ID DUPLICATES
COMPUTE SUM OF SALARY ON DEPARTMENT_ID
COMPUTE AVG OF SALARY ON JOB_ID
SELECT DEPARTMENT_ID, JOB_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE JOB_ID IN ('SH_CLERK', 'SA_MAN')
```

```
AND DEPARTMENT_ID IN (50, 80)
ORDER BY DEPARTMENT_ID, JOB_ID;


DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------- ----------
           50 SH_CLERK   Taylor                         3200
              SH_CLERK   Fleaur                         3100
                            .
                            .
                            .
              SH_CLERK   Gates                          2900


DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------- ----------
           50 SH_CLERK   Perkins                        2500
              SH_CLERK   Bell                           4000
                            .
                            .
                            .
              SH_CLERK   Grant                          2600
              **********                           ----------
              avg                                       3215


DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------- ----------

*************                                      ----------
sum                                                   64300

           80 SA_MAN     Russell                       14000
              SA_MAN     Partners                      13500
              SA_MAN     Errazuriz                     12000
              SA_MAN     Cambrault                     11000
              SA_MAN     Zlotkey                       10500
              **********                           ----------
              avg                                      12200


DEPARTMENT_ID JOB_ID     LAST_NAME                    SALARY
------------- ---------- ------------------------- ----------

*************                                      ----------
sum                                                   61000

25 rows selected.
```

13.11 BTITLE

Syntax

```
BTI[TLE] [printspec [text | variable] ...] | [ON | OFF]
```

where *printspec* represents one or more of the following clauses used to place and format the text:

BOLD

CE[NTER]

COL *n*

FORMAT *text*

LE[FT]

R[IGHT]

S[KIP] [*n*]

TAB *n*

Places and formats a specified title at the bottom of each report page, or lists the current BTITLE definition.

Enter BTITLE with no clauses to list the current BTITLE definition. For a description of the old form of BTITLE, see BTI[TLE] text (obsolete old form).

Terms

See the TTITLE command for information on terms and clauses in the BTITLE command syntax.

Usage

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.

SQL*Plus substitution variables (& variables) are expanded before BTITLE is executed. The resulting string is stored as the BTITLE text. During subsequent execution for each page of results, the expanded value of a variable may itself be interpreted as a variable with unexpected results.

You can avoid this double substitution in a BTITLE command by not using the & prefix for variables that are to be substituted on each page of results. If you want to use a substitution variable to insert unchanging text in a BTITLE, enclose it in quotes so that it is only substituted once.

Examples

To set a bottom title with CORPORATE PLANNING DEPARTMENT on the left and a date on the right, enter

```
BTITLE LEFT 'CORPORATE PLANNING DEPARTMENT' -
RIGHT '1 JAN 2001'
```

To set a bottom title with CONFIDENTIAL in column 50, followed by six spaces and a date, enter

```
BTITLE COL 50 'CONFIDENTIAL'
TAB 6
'1 JAN 2001'
```

13.12 CHANGE

Syntax

```
C[HANGE] sepchar old [sepchar [new [sepchar]]]
```

Changes the first occurrence of the specified text on the current line in the buffer.

Terms

sepchar

Represents any non-alphanumeric character such as "/" or "!". Use a *sepchar* that does not appear in *old* or *new*.

old

Represents the text you wish to change. CHANGE ignores case in searching for *old*. For example,

```
CHANGE /aq/aw
```

finds the first occurrence of "aq", "AQ", "aQ", or "Aq" and changes 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

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 Database error. SQL*Plus sets the buffer's current line to the line containing the error so that you can make modifications.

To reenter 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, SQL*Plus inserts the line at the beginning of the buffer (that line becomes line 1).

Examples

Enter 3 so the current line of the buffer contains the following text:

```
3
```

```
3* WHERE JOB_ID IS IN ('CLERK', 'SA_MAN')
```

Enter the following command:

```
CHANGE /CLERK/SH_CLERK/
```

The text in the buffer changes as follows:

```
3* WHERE JOB_ID IS IN ('SH_CLERK', 'SA_MAN')
```

Or enter the following command:

```
CHANGE /'CLERK',... /'SH_CLERK'/
```

The original line changes to

```
3* WHERE JOB_ID IS IN ('SH_CLERK')
```

Or enter the following command:

```
CHANGE /(...)/('SA_MAN')/
```

The original line changes to

```
3* WHERE JOB_ID IS IN ('SA_MAN')
```

You can replace the contents of an entire line using the line number. This entry

```
3  WHERE JOB_ID IS IN ('SH_CLERK')
```

causes the second line of the buffer to be replaced with

```
WHERE JOB_ID IS IN ('SH_CLERK')
```

Note that entering a line number followed by a string will replace the line regardless of what text follows the line number. For example,

```
2  CHANGE/OLD/NEW/
```

will change the second line of the buffer to be

```
2* CHANGE/OLD/NEW/
```

13.13 CLEAR

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]
```

Resets or erases the current value or setting for the specified option.

Terms

```
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 SET BUF[FER] {buffer|SQL} (obsolete) for more information about the obsolete form of this command.

```
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 SET BUF[FER] {buffer|SQL} (obsolete) for more information about the obsolete form of this command.

```
TIMI[NG]
```

Deletes all timers created by the TIMING command.

Examples

To clear breaks, enter

```
CLEAR BREAKS
```

To clear column definitions, enter

```
CLEAR COLUMNS
```

13.14 COLUMN

Syntax

```
COL[UMN] [{column | expr} [option ...]]
```

where *option* represents one of the following clauses:

```
ALI[AS] alias BOOL[EAN]{YES [NO]} CLE[AR] ENTMAP {ON | OFF} FOLD_A[FTER]
FOLD_B[EFORE] FOR[MAT] format HEA[DING] text JUS[TIFY] {L[EFT] | C[ENTER] |
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]
```

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

- Change the default BOOLEAN value

It also lists the current display attributes for a single column or for all columns.

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.

Terms

{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 LAST_NAME applies to all columns named LAST_NAME 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.

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.

ENTMAP {ON | OFF}

Enables entity mapping to be turned ON or OFF for selected columns in HTML output. This feature enables you to include, for example, HTML hyperlinks in a column of data, while still mapping entities in other columns of the same report. By turning entity

mapping `OFF` for a column containing HTML hyperlinks, the HTML anchor tag delimiters, `<`, `>`, ", and, `&` are correctly interpreted in the report. Otherwise, they would be replaced with their respective entities, `<`, `>`, `"` and `&`, preventing web browsers from correctly interpreting the HTML.

Entities in the column heading and any `COMPUTE` labels or output appearing in the column are mapped or not mapped according to the value of `ENTMAP` for the column.

The default setting for `COLUMN ENTMAP` is the current setting of the `MARKUP HTML ENTMAP` option.

For more information about the `MARKUP HTML ENTMAP` option, see `SET` MARKUP Options.

`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_A[FTER]` does not work in `SET MARKUP HTML ON` mode unless `PREFORMAT` is set `ON`.

`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. `FOLD_A[FTER]` does not work in `SET MARKUP HTML ON` mode unless `PREFORMAT` is set `ON`.

`FOR[MAT]` *format*

Specifies the display format of the column. The format specification must be a text constant, such as A10 or $9,999.

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 data types 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`, `BLOB`, `BFILE`, `CLOB`, `NCLOB`, `XMLType`, or `JSON` column's width defaults to the value of `SET LONGCHUNKSIZE` or `SET LONG`, whichever is smaller.

To change the width of a data type to *n*, use `FORMAT` A*n*. (A stands for alphabetic.) If you specify a width shorter than the column heading, SQL*Plus truncates the heading.

SQL*Plus truncates or wraps `XMLType` columns after 2000 bytes. To avoid this, you need to set an explicit `COLUMN` format for the `XMLType` column. A `COLUMN` format can be up to a maximum of 60,000 per row.

DATE Columns

The default width and format of unformatted `DATE` columns in SQL*Plus are derived from the `NLS_DATE_FORMAT` parameter. The `NLS_DATE_FORMAT` setting is determined by the NLS territory parameter. For example, the default format for the NLS territory, America, is `DD-Mon-RR`, and the default width is `A9`. The NLS parameters may be set in your database parameter file, in environment variables, or an equivalent platform-specific mechanism. They may also be specified for each session with the `ALTER SESSION` command. For more information about `DATE` formats and about NLS parameters, see the *Oracle Database SQL Language Reference*.

You can change the format of the `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 Database automatically enables a very wide column. The default column width may also depend on the character sets in use in SQL*Plus and in the database. To maximize script portability if multiple character sets are used, Oracle Database recommends using `COLUMN FORMAT` for each column selected.

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

For numeric columns, `COLUMN FORMAT` settings take precedence over `SET NUMFORMAT` settings, which take precedence over `SET NUMWIDTH` settings.

See SET NUMF[ORMAT] format and SET NUM[WIDTH] {10 | n}.

To change a `NUMBER` column's width, use `FORMAT` followed by an element as specified in Table 13-1.

Table 13-1   Number Formats

Element	Examples	Description
, (comma)	9,999	Displays a comma in the specified position.
. (period)	99.99	Displays a period (decimal point) to separate the integral and fractional parts of a number.
$	$9999	Displays a leading dollar sign.
0	0999 9990	Displays leading zeros Displays trailing zeros.
9	9999	Displays a value with the number of digits specified by the number of 9s. Value has a leading space if positive, a leading minus sign if negative. Blanks are displayed for leading zeroes. A zero (0) is displayed for a value of zero.
B	B9999	Displays blanks for the integer part of a fixed-point number when the integer part is zero, regardless of zeros in the format model.
C	C999	Displays the ISO currency symbol in the specified position.
D	99D99	Displays the decimal character to separate the integral and fractional parts of a number.
EEEE	9.999EEEE	Displays value in scientific notation (format must contain exactly four "E"s).
G	9G999	Displays the group separator in the specified positions in the integral part of a number.
L	L999	Displays the local currency symbol in the specified position.

Table 13-1 (Cont.) Number Formats

Element	Examples	Description
MI	9999MI	Displays a trailing minus sign after a negative value. Display a trailing space after a positive value.
PR	9999PR	Displays a negative value in <angle brackets>. Displays a positive value with a leading and trailing space.
RN rn	RN rn	Displays uppercase Roman numerals. Displays lowercase Roman numerals. Value can be an integer between 1 and 3999.
S	S9999 9999S	Displays a leading minus or plus sign. Displays a trailing minus or plus sign.
TM	TM	Displays the smallest number of decimal characters possible. The default is TM9. Fixed notation is used for output up to 64 characters, scientific notation for more than 64 characters. Cannot precede TM with any other element. TM can only be followed by a single 9 or E
U	U9999	Displays the dual currency symbol in the specified position.
V	999V99	Displays value multiplied by 10n, where n is the number of 9's after the V.
X	XXXX xxxx	Displays the hexadecimal value for the rounded value of the specified number of digits.

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 COLUMN FORMAT or SET NUMFORMAT, then the column's width will always be at least the value of SET NUMWIDTH.

SQL*Plus may round your NUMBER data to fit your format or field width.

If a value cannot fit in the column, SQL*Plus displays pound signs (#) instead of the number.

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 (-~).

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 LAST_NAME HEADING 'Employee |Name'
```

would produce a two-line column heading.

See SET HEADS[EP] { | c | ON | OFF} for information on changing the `HEADSEP` character.

```
JUS[TIFY] {L[EFT] | C[ENTER] | 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`. `NEWL[INE]` does not work in `SET MARKUP HTML ON` mode unless `PREFORMAT` is SET `ON`.

```
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 primary/detail reports in which there is a new primary record for each page. For primary/detail reporting, you must also include the column in the `ORDER BY` clause. See the example at the end of this command description.

Variables specified with `NEW_V[ALUE]` are expanded before `TTITLE` is executed. The resulting string is stored as the `TTITLE` text. During subsequent execution for each page of the report, the expanded value of a variable may itself be interpreted as a variable with unexpected results.

You can avoid this double substitution in a `TTITLE` command by not using the `&` prefix for `NEW_V[ALUE]` variables that are to be substituted on each page of the report. If you want to use a substitution variable to insert unchanging text in a `TTITLE`, enclose it in quotes so that it is only substituted once.

For information on displaying a column value in the bottom title, see `OLD_V[ALUE]` variable below. For more information on referencing variables in titles, see the TTITLE command. For information on formatting and valid format models, see `FOR[MAT]` format above.

```
NOPRI[NT] | PRI[NT]
```

Controls the printing of the column (the column heading and all the selected values). NOPRINT turns off the screen output and printing of the column. 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 always becomes 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 primary/detail reports in which there is a new primary record for each page. For primary/detail reporting, you must also include the column in the ORDER BY clause.

Variables specified with OLD_V[ALUE] are expanded before BTITLE is executed. The resulting string is stored as the BTITLE text. During subsequent execution for each page of the report, the expanded value of a variable may itself be interpreted as a variable with unexpected results.

You can avoid this double substitution in a BTITLE command by not using the & prefix for OLD_V[ALUE] variables that are to be substituted on each page of the report. If you want to use a substitution variable to insert unchanging text in a BTITLE, enclose it in quotes so that it is only substituted once.

For information on displaying a column value in the top title, see the NEW_V[ALUE] variable. For more information on referencing variables in titles, see the TTITLE command.

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 data type 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.

NCLOB, BLOB, BFILE, or multibyte CLOB columns cannot be formatted with the WORD_WRAPPED option. If you format an NCLOB, BLOB, BFILE, or multibyte CLOB column with COLUMN WORD_WRAPPED, the column data behaves as though COLUMN WRAPPED was applied instead.

BOOLEAN Columns

Starting with Release 23ai, the COLUMN command for BOOLEAN data type sets the output format for the values returned from the database to a different text literal other than TRUE or FALSE.

Syntax

```
COLUMN column BOOLEAN TEXT1 [TEXT2]
```
where,

TEXT1 is the text to represent the BOOLEAN value TRUE returned from the database. It is a mandatory value.

TEXT2 is an optional value to represent the BOOLEAN value FALSE returned from the database. When TEXT2 is not provided, the value returned from the database is displayed.

For more information about BOOLEAN values, see the *Oracle Database SQL Language Reference*.

Usage

The COLUMN commands you enter can control a column's display attributes for multiple SQL SELECT commands.

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.

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

You can make the LAST_NAME column 20 characters wide and display EMPLOYEE NAME on two lines as the column heading:

```
COLUMN LAST_NAME FORMAT A20 HEADING 'EMPLOYEE|NAME'
```

You can format the SALARY 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:

```
COLUMN SALARY FORMAT $9,999,990.99
```

You can assign the alias NET to a column containing a long expression, display the result in dollar format, and display <NULL> for null values:

```
COLUMN SALARY+COMMISSION_PCT+BONUS-EXPENSES-INS-TAX ALIAS NET
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 in the SELECT command. Otherwise, SQL*Plus cannot match the COLUMN command to the appropriate column.

You can wrap long values in a column named REMARKS as follows:

```
COLUMN REMARKS FORMAT A20 WRAP
```

```
CUSTOMER    DATE       QUANTITY REMARKS
----------  ---------  -------- --------------------
123         25-AUG-2001 144      This order must be s
```

```
                                   hipped by air freigh
                                   t to ORD
```

If you replace WRAP with WORD_WRAP, the REMARKS column looks like this:

```
CUSTOMER      DATE          QUANTITY    REMARKS
----------    ---------     --------    ---------------------
123           25-AUG-2001 144           This order must be
                                        shipped by air freight
                                        to ORD
```

If you specify TRUNCATE, the REMARKS column looks like this:

```
CUSTOMER      DATE          QUANTITY    REMARKS
----------    ---------     --------    --------------------
123           25-AUG-2001 144           This order must be s
```

To print the current date and the name of each job in the top title, enter the following. Use the EMPLOYEES table of the HR schema instead of EMP_DETAILS_VIEW.

For details on creating a date variable, see About Displaying the Current Date in Titles.

Your two-page report would look similar to the following report, with "Job Report" centered within your current linesize:

```
COLUMN JOB_ID NOPRINT NEW_VALUE JOBVAR
COLUMN TODAY  NOPRINT NEW_VALUE DATEVAR
BREAK ON JOB_ID SKIP PAGE ON TODAY
TTITLE CENTER 'Job Report' RIGHT DATEVAR  SKIP 2 -
LEFT 'Job:     ' JOBVAR SKIP 2
SELECT TO_CHAR(SYSDATE, 'MM/DD/YYYY') TODAY,
LAST_NAME, JOB_ID, MANAGER_ID, HIRE_DATE, SALARY, DEPARTMENT_ID
FROM EMPLOYEES WHERE JOB_ID IN ('MK_MAN', 'SA_MAN')
ORDER BY JOB_ID, LAST_NAME;
```

You can change the default format of DATE columns to YYYY-MM-DD:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD';
```

```
Session altered.
```

Execute the SELECT statement to display the change:

```
SELECT HIRE_DATEFROM EMPLOYEESWHERE EMPLOYEE_ID = 206;
```

The following output is displayed:

```
Job Report                             04/19/01

Job:     SA_MAN

HIRE_DATE
```

```
----------
1994-06-07
```

See ALTER SESSION for information on the `ALTER SESSION` command.

You can change the default `BOOLEAN` value for `COL1` from `TRUE` and `FALSE` to `YES` and `NO`:

```
SET NULL BLANK
COLUMN COL1 BOOLEAN YES NO
```

```
SELECT * FROM my_table;
```

The following output is displayed:

```
ID COL1  COL2
1 YES    BLANK
2 NO     TRUE
3 YES    TRUE
4 NO     TRUE
5 NO     TRUE
6 NO     TRUE
7 YES    FALSE
8 YES    BLANK
```

13.15 COMPUTE

Syntax

```
COMP[UTE] [function [LAB[EL] text] ...    OF {expr | column | alias} ...     ON
{expr | column | alias | REPORT | ROW} ...]
```

In combination with the BREAK command, calculates and prints summary lines, using various standard computations on subsets of selected rows. It also lists all COMPUTE definitions. For details on how to create summaries, see **About Clarifying Your Report with Spacing and Summary Lines**.

Terms

function ...

Represents one of the functions listed in Table 13-2. If you specify more than one function, use spaces to separate the functions.

COMPUTE command functions are always executed in the sequence AVG, COUNT, MINIMUM, MAXIMUM, NUMBER, SUM, STD, VARIANCE, regardless of their order in the COMPUTE command.

Table 13-2   COMPUTE Functions

Function	Computes	Applies to Datatypes
`AVG`	Average of non-null values	NUMBER
`COU[NT]`	Count of non-null values	all types
`MIN[IMUM]`	Minimum value	NUMBER, CHAR, NCHAR, VARCHAR2 (VARCHAR), NVARCHAR2 (NCHAR VARYING)
`MAX[IMUM]`	Maximum value	NUMBER, CHAR, NCHAR, VARCHAR2 (VARCHAR), NVARCHAR2 (NCHAR VARYING)
`NUM[BER]`	Count of rows	all types
`SUM`	Sum of non-null values	NUMBER
`STD`	Standard deviation of non-null values	NUMBER
`VAR[IANCE]`	Variance of non-null values	NUMBER

`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. You must place single quotes around *text* containing spaces or punctuation. The label prints left justified and truncates to the column width or linesize, whichever is smaller. The maximum label length 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*`} ...`

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} ...`

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

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.

Note that if you use the NOPRINT option for the column on which the COMPUTE is being performed, the COMPUTE result is also suppressed.

Examples

To subtotal the salary for the "account manager", AC_MGR, and "salesman", SA_MAN, job classifications with a compute label of "TOTAL", enter

```
BREAK ON JOB_ID SKIP 1;
COMPUTE SUM LABEL 'TOTAL' OF SALARY ON JOB_ID;
SELECT JOB_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE JOB_ID IN ('AC_MGR', 'SA_MAN')
ORDER BY JOB_ID, SALARY;


JOB_ID     LAST_NAME                    SALARY
---------- ------------------------- ----------
AC_MGR     Higgins                        12000
**********                           ----------
TOTAL                                     12000
SA_MAN     Zlotkey                        10500
           Cambrault                      11000
           Errazuriz                      12000
           Partners                       13500
           Russell                        14000
**********                           ----------
TOTAL                                     61000

6 rows selected.
```

To calculate the total of salaries greater than 12,000 on a report, enter

```
COMPUTE SUM OF SALARY ON REPORT
BREAK ON REPORT
COLUMN DUMMY HEADING ''
SELECT '    ' DUMMY, SALARY, EMPLOYEE_ID
FROM EMP_DETAILS_VIEW
```

```
WHERE SALARY > 12000
ORDER BY SALARY;


        SALARY EMPLOYEE_ID
--- ---------- -----------
         13000         201
         13500         146
         14000         145
         17000         101
         17000         102
         24000         100
    ----------
sum     98500


6 rows selected.
```

To calculate the average and maximum salary for the executive and accounting departments, enter

```
BREAK ON DEPARTMENT_NAME SKIP 1
COMPUTE AVG LABEL 'Dept Average' -
        MAX LABEL 'Dept Maximum' -
        OF SALARY ON DEPARTMENT_NAME
SELECT DEPARTMENT_NAME, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE DEPARTMENT_NAME IN ('Executive', 'Accounting')
ORDER BY DEPARTMENT_NAME;


DEPARTMENT_NAME                 LAST_NAME                      SALARY
------------------------------  -------------------------  ----------
Accounting                      Higgins                         12000
                                Gietz                            8300
****************************                               ----------
Dept Average                                                    10150
Dept Maximum                                                    12000

Executive                       King                            24000
                                Kochhar                         17000
                                De Haan                         17000
****************************                               ----------
Dept Average                                               19333.3333
Dept Maximum                                                    24000
```

To sum salaries for departments <= 20 without printing the compute label, enter

```
COLUMN DUMMY NOPRINT
COMPUTE SUM OF SALARY ON DUMMY
BREAK ON DUMMY SKIP 1
SELECT DEPARTMENT_ID DUMMY, DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE DEPARTMENT_ID <= 20
ORDER BY DEPARTMENT_ID;


DEPARTMENT_ID LAST_NAME                     SALARY
------------- ------------------------- ----------
```

```
          10 Whalen                              4400
                                           ----------
                                                 4400

          20 Hartstein                          13000
          20 Fay                                 6000
                                           ----------
                                                19000
```

To total the salary at the end of the report without printing the compute label, enter

```
COLUMN DUMMY NOPRINT
COMPUTE SUM OF SALARY ON DUMMY
BREAK ON DUMMY
SELECT NULL DUMMY, DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE DEPARTMENT_ID <= 30
ORDER BY DEPARTMENT_ID;


DEPARTMENT_ID LAST_NAME                      SALARY
------------- -------------------------- ----------
          10 Whalen                           4400
          20 Hartstein                       13000
          20 Fay                              6000
          30 Raphaely                        11000
          30 Khoo                             3100
          30 Baida                            2900
          30 Tobias                           2800
          30 Himuro                           2600
          30 Colmenares                       2500
                                          ----------
                                               48300
```

```
9 rows selected.
```

13.16 CONFIG

Syntax

```
CONFIG EXPORT TNS [<tnsnames.ora file location>] [ AZURE ] [ [ FILE ] [<JSON
filename>] ]
```

Generates the Centralized Configuration Provider JSON file from the `tnsnames.ora` file located at the default path. The default file name, `oraconfig.json`, will be used if you do not specify the name of the JSON file. You can override the default path of the `tnsnames.ora` file by providing it explicitly in this command.

By default, the `CONFIG` command generates the Centralized Configuration Provider JSON file for OCI cloud service. It also generates Centralized Configuration Provider JSON file for Azure cloud service.

Terms

```
tnsnames.ora file location
```

Path of the `tnsnames.ora` file. This path overrides the default location of the `tnsnames.ora` file in the environment.

```
AZURE
```

Generates the Centralized Configuration Provider JSON file for Oracle Database Service for Azure.

```
FILE [JSON filename]
```

Stores the generated JSON content in a file. If you do not specify the file name, the JSON file is created with the default file name, which is `oraconfig.json`. Otherwise, the file name provided with the `FILE` clause is used.

> **Note:**
>
> To generate the JSON file, the `CONFIG` command must include either the file name or the `FILE` keyword. When both the file name and the `FILE` keyword are not specified, the generated JSON content is displayed on the console only.

Examples

Execute the following command to generate a JSON file in the default directory using the default file name from the `tnsnames.ora` file:

```
CONFIG EXPORT TNS FILE
```

The following output shows the successful generation of the JSON file:

```
Generating config store JSON for Local Net Naming configuration file /opt/
oracle/tnsnames.ora
Config store JSON file generated successfully (/home/orcl/oraconfig.json)
```

The following output shows the content of the JSON file:

```
 {
    "cdb1_pdb1": {
        "connect_descriptor":
"(DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=cdb1_pdb1))(ADDRESS=(PROTOCOL=tcp)
(HOST=host1.customer.domain.com)(PORT=1521)))"
    }
}
```

Execute the following command to generate a JSON file for Oracle Database Service for Azure in the default directory using the default file name from the `tnsnames.ora` file:

```
 CONFIG EXPORT TNS AZURE FILE
```

The following output shows the successful generation of the JSON file:

```
Generating config store JSON for Local Net Naming configuration
file /opt/oracle/tnsnames.ora
Config store JSON file generated successfully (/home/orcl/
oraconfig.json)
```

The following output shows the content of the `oraconfig.json` file:

```
{
"orcl/cdb1_pdb1/connect_descriptor":
"(DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=cdb1_pdb1))
(ADDRESS=(PROTOCOL=tcp)(HOST=host1.customer.domain.com)(PORT=1521)))"}
```

Execute the following command to generate a JSON file with the file name `myconfig.json` from the `tnsnames.ora` file present at the default location:

```
CONFIG EXPORT TNS FILE myconfig.json
```

The following output shows the successful generation of the JSON file:

```
Generating config store JSON for Local Net Naming configuration
file /opt/oracle/tnsnames.ora
Config store JSON file generated successfully (/home/orcl/
oraconfig.json)
```

Execute the following command to generate a JSON file with the file name `mynewconfig.json` from the `tnsnames.ora` file that is present at a different location:

```
CONFIG EXPORT TNS c:\temp\tnsnames.ora FILE mynewconfig.json
```

The following output shows the successful generation of the JSON file:

```
Generating config store JSON for Local Net Naming configuration file
c:\temp\tnsnames.ora
Config store JSON file generated successfully
(C:\Users\USER1\mynewconfig.json)
```

Execute the following command to generate JSON content and display it on the console:

```
CONFIG EXPORT TNS
```

The following shows the JSON content:

```
Generating config store JSON for Local Net Naming configuration
file /opt/oracle/tnsnames.ora

{
    "cdb1_pdb1": {
```

```
        "connect_descriptor":
"(DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=cdb1_pdb1))(ADDRESS=(PROTOCOL=tcp)
(HOST=host1.customer.domain.com)(PORT=1521)))"
    }
}
```

13.17 CONNECT

Syntax

```
CONN[ECT] [{logon | / | proxy}  [AS {SYSASM |SYSBACKUP |SYSDBA |SYSDG |SYSOPER |
SYSRAC | SYSKM}]  [edition=value]]
```

where *logon* has the syntax:

```
username[/password] [@connect_identifier]
```

where *proxy* has the syntax:

```
proxyuser[username] [/password] [@connect_identifier]
```

> **Note:**
>
> The brackets around *username* in *proxy* are required syntax, not an indication of an optional term. For example, to connect to scott through proxy user hr with password welcome1.
>
> ```
> CONNECT hr[scott]/welcome1
> ```

Connects a given username to the Oracle Database. When you run a CONNECT command, the site profile, glogin.sql, and the user profile, login.sql, are executed.

CONNECT does not reprompt for username or password if the initial connection does not succeed.

> **WARNING:**
>
> Including your password in plain text is a security risk. You can avoid this risk by omitting the password, and entering it only when the system prompts for it.

To connect to a database using an enterprise user proxy, you must first configure the proxy. For information about configuring an enterprise user proxy, see the *Oracle Database Enterprise User Security Administrator's Guide*.

Terms

```
username[/password]
```

The username and password you use to connect to Oracle Database. If you omit *username* and *password*, SQL*Plus prompts you for them. If you enter a slash (/) or enter Return or

click Execute when prompted for *username*, SQL*Plus logs you in using a default logon. See / (slash) for more information.

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 for information about changing your password in SQL*Plus.

```
connect_identifier
```

An Oracle Net connect identifier. The exact syntax depends on the Oracle Net configuration. For more information, refer to the Oracle Net manual or contact your DBA. SQL*Plus does not prompt for a service name, but uses your default database if you do not include a connect identifier.

A connect_identifier is also used to connect to a pluggable database (PDB). See *Oracle Database Administrator's Guide*

```
edition=value
```

The value for the Oracle Session Edition. An edition enables two or more versions of an object in a database. It provides a staging area where changed objects can be loaded into the database, compiled, and executed during uptime. This is particularly useful to reduce downtime associated with patching an application. *edition=value* overrides any edition value specified in the ORA_EDITION environment variable. For more detailed information, see *Oracle Database Administrator's Guide*.

```
/ (slash)
```

Represents a default logon using operating system authentication. You cannot enter a connect_identifier 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 *Oracle Database Administrator's Guide* for information about operating system authentication.

In SQL*Plus command line, where applications use password credentials to connect to databases, it is possible to store the credentials in a client-side Oracle wallet. When you configure a client to use the Oracle wallet, applications can use the following syntax to connect to databases that use password authentication:

```
CONNECT /@database_alias
```

For information about configuring your client to use secure external password store and for information about managing credentials in it, see the *Oracle Database Security Guide*.

```
AS {SYSASM |SYSBACKUP |SYSDBA |SYSDG |SYSOPER |SYSRAC |SYSKM}
```

The AS clause enables privileged connections by users who have been granted that system privileges. You can use any one of these privileged connections with the default logon, /.

For information about system privileges, see the *Oracle Database Administrator's Guide*.

Usage

CONNECT commits the current transaction to the database, disconnects the current username from Oracle Database, 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 more information about user account management, refer to the CREATE USER, ALTER USER and the CREATE PROFILE commands in the *Oracle Database SQL Language Reference*.

Examples

To connect across Oracle Net with username HR, to the database known by the Oracle Net alias as FLEETDB, enter

```
CONNECT HR@FLEETDB
```

For more information about setting up your password file, refer to the *Oracle Database Administrator's Guide*.

To connect to an instance on the current node as a privileged user named HR, enter

```
CONNECT HR AS SYSDBA
```

To connect to an instance on the current node as a privileged default user, enter

```
CONNECT / AS SYSDBA
```

You can use the CONNECT command to connect to a CDB using easy connect or a net service name.

This statement connects to the **hr** user using the **hrapp** service. The **hrapp** service has a PDB property for the **hrpdb** PDB. This example assumes that the client is configured to have a **Net Service Name** for the **hrapp** service.

```
CONNECT hr@hrapp
```

13.18 COPY

The COPY command is not being enhanced to handle datatypes or features introduced with, or after Oracle8*i*. The COPY command is likely to be deprecated in a future release.

For COPY command details and syntax, see SQL*Plus COPY Command.

13.19 DEFINE

Syntax

```
DEF[INE] [variable] | [variable = text]
```

Specifies a user or predefined variable and assigns a CHAR value to it, or lists the value and variable type of a single variable or all variables.

Terms

```
variable
```

Represents the user or predefined 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 substitution 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 substitution variables.

Usage

Defined variables retain their values until you:

- enter a new DEFINE command referencing the variable
- enter an UNDEFINE command referencing the variable
- enter an ACCEPT command referencing the variable
- reference the variable in the NEW_VALUE or OLD_VALUE clause of a COLUMN command and then reference the column in a SELECT command
- EXIT SQL*Plus

Whenever you run a stored query or script, 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*.

If the value of a defined variable extends over multiple lines (using the SQL*Plus command continuation character), SQL*Plus replaces each continuation character and carriage return with a space. For example, SQL*Plus interprets

```
DEFINE TEXT = 'ONE-
TWO-
THREE'
```

as

```
DEFINE TEXT = 'ONE TWO THREE'
```

You should avoid defining variables with names that may be identical to values that you will pass to them, as unexpected results can occur. If a value supplied for a defined variable matches a variable name, then the contents of the matching variable are used instead of the supplied value.

Some variables are predefined when SQL*Plus starts. Enter DEFINE to see their definitions.

Examples

To assign the value MANAGER to the variable POS, type:

```
DEFINE POS = MANAGER
```

If you execute a command containing a reference to &POS, SQL*Plus substitutes the value MANAGER for &POS and will not prompt you for a POS value.

To assign the CHAR value 20 to the variable DEPARTMENT_ID, type:

```
DEFINE DEPARTMENT_ID = 20
```

Even though you enter the number 20, SQL*Plus assigns a CHAR value to DEPARTMENT_ID consisting of two characters, 2 and 0.

To list the definition of DEPARTMENT_ID, enter

```
DEFINE DEPARTMENT_ID
```

```
DEFINE DEPARTMENT_ID = "20" (CHAR)
```

This result shows that the value of DEPARTMENT_ID is 20.

13.19.1 Predefined Variables

There are nine variables defined during SQL*Plus installation. These variables only differ from user-defined variables by having predefined values.

Table 13-3   Variables Predefined at SQL*Plus Installation

Variable Name	Contains
_CONNECT_IDENTIFIER	Connection identifier used to make connection, where available.
_DATE	Current date, or a user defined fixed string.
_EDITOR	Specifies the editor used by the EDIT command.
_O_VERSION	Current version of the installed Oracle Database.
_O_RELEASE	Full release number of the installed Oracle Database.
_PRIVILEGE	Privilege level of the current connection.
_SQLPLUS_RELEASE	Full release number of installed SQL*Plus component.
_USER	User name used to make connection.
_SQL_ID	sql_id of the SQL statement executed.

_CONNECT_IDENTIFIER

Contains the `INSTANCE_NAME,` `SERVICE_NAME` or `ORACLE_SID` from the connection identifier. If a connection identifier is not supplied by the user during connection, the `_CONNECT_IDENTIFIER` contains the `ORACLE_SID`.

_DATE

Contains either the current date as a dynamic variable, or a fixed string. The current date is the default and is formatted using the value of NLS_DATE_FORMAT.

Because _DATE can be used as a normal substitution variable, users may put it in TTITLE. If _DATE is dynamic and is used in TTITLE it will have all the normal variable semantics. If it is used with an ampersand than the value will be set to the time when the TTITLE command is executed. If it is used without an ampersand prefix, it will be re-evaluated for each page. For long reports with _DATE in the TTITLE or with multiple references to &_DATE, different times may be displayed for each occurrence of the variable.

Users using _DATE in TTITLEs will almost certainly want to use an ampersand: &_DATE, so that each page of the report has exactly the same timestamp. This is especially true when the current date format contains a "seconds" component.

A DEFINE (with no arguments) or dereference using &_DATE will give the current date.

The _DATE value can be UNDEFINED, or set to a fixed string with an explicit DEFINE _DATE.

You can re-enable the default dynamic date behavior with:

```
DEFINE _DATE = "" (an empty string)
```

_DATE enables time values to be included in your SQL*Plus prompt.

_EDITOR

Specifies the default editor used by the EDIT command.

During SQL*Plus installation on Windows operating systems, it is set to Notepad. On UNIX operating systems, it is set to the value of the UNIX environment variable, EDITOR, if it exists, otherwise it is set to Ed.

You can use the DEFINE command to redefine _EDITOR, to hold the name of your preferred text editor. For example, to define the editor used by EDIT to be vi, enter the following command:

```
DEFINE _EDITOR = vi
```

_O_VERSION

Contains the current version of the installed Oracle Database.

_O_RELEASE

Contains the full release number of the installed Oracle Database in the form:

1801000000

_PRIVILEGE

Contains a value indicating the privilege level of the current connection. It contains one of the following values:

- AS SYSASM
- AS SYSBACKUP
- AS SYSDBA
- AS SYSDG
- AS SYSOPER
- AS SYSRAC
- An empty string for normal-user connections or when there is no connection.

AS SYSASM, AS SYSBACKUP, AS SYSDBA, AS SYSDG, AS SYSOPER and AS SYSRAC are database administrator level privileges.

> **See Also:**
>
> GRANT for information on AS SYSDBA and AS SYSOPER privileges.

_SQLPLUS_RELEASE

Contains the full release number of the installed SQL*Plus component in the form:

1801000000

_USER

Contains the user name connected to the current connection.

_SQL_ID

Contains the sql_id for the currently executed SQL or PL/SQL statements.

You can view the value of each of these variables with the DEFINE command.

These variables can be accessed and redefined like any other substitution variable. They can be used in TTITLE, in '&' substitution variables, or in your SQL*Plus command-line prompt.

You can use the DEFINE command to view the definitions of these nine predefined variables in the same way as you view other DEFINE definitions. You can also use the DEFINE command to redefine their values, or you can use the UNDEFINE command to remove their definitions and make them unavailable.

To view a specific variable definition, enter

```
DEFINE variable
```

where *variable* is the name of the substitution variable whose definition you want to view.

To view all predefined and user defined variable definitions, enter

```
DEFINE
```

All predefined and all user defined variable definitions are displayed.

You can use UNDEFINE to remove a substitution variable definition and make it unavailable.

Examples of Use of Predefined Variables

To change your SQL*Plus prompt to display your connection identifier, enter:

```
SET SQLPROMPT '_CONNECT_IDENTIFIER > '
```

To view the predefined value of the _SQLPLUS_RELEASE substitution variable, enter

```
DEFINE _SQLPLUS_RELEASE
```

The value of the predefined variable _SQLPLUS_RELEASE is displayed.

```
DEFINE _SQLPLUS_RELEASE = "1801000000" (CHAR)
```

13.20 DESCRIBE

Syntax

```
DESC[RIBE] {[schema.]object[@db_link]}
```

Lists the column definitions for the specified table, view or synonym, or the specifications for the specified function or procedure.

Terms

schema

Represents the schema where the object or permission to describe the object resides. If you omit *schema* and the object is not a *public synonym*, then the currently available *schema* is used.

object

Represents the table, view, type, procedure, function, package, or synonym you wish to describe.

@db_link

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 *Oracle Database SQL Language Reference*.

Usage

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

- Data type of columns, for example, `CHAR`, `DATE`, `LONG`, `LONGRAW`, `NUMBER`, `RAW`, `ROWID`, `VARCHAR2 (VARCHAR)`, `XMLType`, `BOOLEAN`

> **✎ Note:**
>
> Starting with Oracle Database 23ai, the `DESCRIBE` command also displays the Domain information (if it exists) associated with the column.
>
> ```
> Name               Null?   Type
> -----------------  -----   -----------------------------
> CUST_EMAIL                 VARCHAR2(100) DOMAIN EMAIL
> ```

- Precision of columns (and scale, if any, for a numeric column)

- Annotation information for a table or view and its columns (if enabled)

> **Note:**
>
> — When annotation is enabled, the `DESCRIBE` command displays the annotation information for a table or view and its columns.
>
> — When annotation is disabled, there is no change to the current behavior. The annotation information is not displayed for a table or view and its columns.
>
>   See Examples.

When you execute the `DESCRIBE` command, the `VARCHAR` columns are returned with a type of `VARCHAR2`.

The `DESCRIBE` command enables you to describe objects recursively to the depth level set in the `SET DESCRIBE` command. You can also display the line number and indentation of the attribute or column name when an object contains multiple object types. For more information, see the `SET` command.

To control the width of the data displayed, use the `SET LINESIZE` command.

Columns output for the `DESCRIBE` command are typically allocated a proportion of the `linesize` currently specified. Decreasing or increasing the `linesize` with the `SET LINESIZE` command usually makes each column proportionally smaller or larger. This may cause unexpected text wrapping in your display. For more information, see the `SET` command.

To enable or disable the display of annotation information, you can use the `SET DESCRIBE` command. With the new `SET DESCRIBE` option, if enabled, the `DESCRIBE` command displays the column metadata of a table or view with its annotations. For more information, see the `SET` command.

The `DESCRIBE` command can be used to retrieve the metadata for the `BOOLEAN` data type.

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 input or output, and default values, if any

* the `ENCRYPT` keyword to indicate whether or not data in a column is encrypted

Examples

You can describe the `EMP_DETAILS_VIEW` view as shown in the following example:

```
DESCRIBE EMP_DETAILS_VIEW
```

The following output is displayed:

```
Name                              Null?    Type
--------------------------------- -------- ----------------
EMPLOYEE_ID                       NOT NULL NUMBER(6)
JOB_ID                            NOT NULL VARCHAR2(10)
MANAGER_ID                                 NUMBER(6)
```

```
DEPARTMENT_ID                                         NUMBER(4)
LOCATION_ID                                           NUMBER(4)
COUNTRY_ID                                            CHAR(2)
FIRST_NAME                                            VARCHAR2(20)
LAST_NAME                                    NOT NULL VARCHAR2(25)
SALARY                                                NUMBER(8,2)
COMMISSION_PCT                                        NUMBER(2,2)
DEPARTMENT_NAME                              NOT NULL VARCHAR2(30)
JOB_TITLE                                    NOT NULL VARCHAR2(35)
CITY                                         NOT NULL VARCHAR2(30)
STATE_PROVINCE                                        VARCHAR2(25)
COUNTRY_NAME                                          VARCHAR2(40)
REGION_NAME                                           VARCHAR2(25)
```

You can describe the CUSTOMER_LOOKUP procedure as shown in the following example:

```
DESCRIBE customer_lookup
```

The following output is displayed:

```
PROCEDURE customer_lookup
Argument Name          Type      In/Out   Default?
---------------------  --------  -------- ---------
CUST_ID                NUMBER    IN
CUST_NAME              VARCHAR2  OUT
```

The procedure MYPROC has a BOOLEAN parameter, as shown in the following example:

```
CREATE PROCEDURE myproc (col1 IN CHAR, col2 IN NUMBER, col3 BOOLEAN) AS
BEGIN
  Null;
END;
/
```

You can describe the procedure MYPROC as shown in the following example:

```
DESCRIBE myproc
```

The following output is displayed:

```
PROCEDURE myproc
Argument Name         Type      In/Out   Default?
------------------  --------  -------- ---------
COL1                 CHAR      IN
COL2                 NUMBER    IN
COL3                 BOOLEAN   IN
```

The package APACK has the procedures APROC and BPROC, as shown in the following example:

```
CREATE PACKAGE apack AS
PROCEDURE aproc(P1 CHAR, P2 NUMBER);
PROCEDURE bproc(P1 CHAR, P2 NUMBER);
```

```
END apack;
/
```

```
Package created.
```

You can describe the package `APACK` as shown in the following example:

```
DESCRIBE apack
```

The following output is displayed:

```
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
```

An object type `ADDRESS` has the attributes `STREET` and `CITY`, as shown in the following example:

```
CREATE TYPE ADDRESS AS OBJECT
  ( STREET  VARCHAR2(20),
    CITY    VARCHAR2(20)
  );
/
```

```
Type created.
```

You can describe the object type `ADDRESS` as shown in the following example:

```
DESCRIBE address
```

The following output is displayed:

```
 Name                          Null?    Type
 ----------------------------- -------- ----------------------
 STREET                                 VARCHAR2(20)
 CITY                                   VARCHAR2(20)
```

An object type `EMPLOYEE` has the attributes `LAST_NAME`, `EMPADDR`, `JOB_ID`, and `SALARY`, as shown in the following code snippet:

```
CREATE TYPE EMPLOYEE AS OBJECT
(LAST_NAME VARCHAR2(30),
EMPADDR ADDRESS,
JOB_ID VARCHAR2(20),
SALARY NUMBER(7,2)
```

```
);
/
```

```
Type created.
```

You can describe the object type EMPLOYEE as shown in the following example:

```
DESCRIBE employee
```

The following output is displayed:

```
Name                             Null?    Type
-------------------------------- -------- ---------------
LAST_NAME                                 VARCHAR2(30)
EMPADDR                                   ADDRESS
JOB_ID                                    VARCHAR2(20)
SALARY                                    NUMBER(7,2)
```

An object type ADDR_TYPE is a table of the object type ADDRESS, as shown in the following example:

```
CREATE TYPE addr_type IS TABLE OF ADDRESS;
/
```

```
Type created.
```

You can describe the object type ADDR_TYPE as shown in the following example:

```
DESCRIBE addr_type
```

The following output is displayed:

```
 addr_type TABLE OF ADDRESS
Name                             Null?    Type
-------------------------------- -------- -------------------
STREET                                    VARCHAR2(20)
CITY                                      VARCHAR2(20)
```

An object type ADDR_VARRAY is a varray of the object type ADDRESS, as shown in the following example:

```
CREATE TYPE addr_varray AS VARRAY(10) OF ADDRESS;
/
```

```
Type created.
```

You can describe the object type ADDR_VARRAY as shown in the following example:

```
DESCRIBE addr_varray
```

The following output is displayed:

```
addr_varray VARRAY(10) OF ADDRESS
Name                             Null?    Type
-------------------------------- -------- -----------------
 STREET                                   VARCHAR2(20)
 CITY                                     VARCHAR2(20)
```

The table DEPARTMENT has the columns DEPARTMENT_ID, PERSON, and LOC, as shown in the following example:

```
CREATE TABLE department
(DEPARTMENT_ID NUMBER,
PERSON EMPLOYEE,
LOC NUMBER
);
/

Table created.
```

You can describe the table DEPARTMENT as shown in the following example:

```
DESCRIBE department
```

The following output is displayed:

```
Name                                  Null?    Type
------------------------------------- -------- --------------------
 DEPARTMENT_ID                                 NUMBER
 PERSON                                        EMPLOYEE
 LOC                                           NUMBER
```

An object type RATIONAL has the attributes NUMERATOR and DENOMINATOR and the method RATIONAL_ORDER, as shown in the following code snippets:

```
CREATE OR REPLACE TYPE rational AS OBJECT
(NUMERATOR NUMBER,
DENOMINATOR NUMBER,
MAP MEMBER FUNCTION rational_order -
RETURN DOUBLE PRECISION,
PRAGMA RESTRICT_REFERENCES
(rational_order, RNDS, WNDS, RNPS, WNPS) );
/

CREATE OR REPLACE TYPE BODY rational AS OBJECT
MAP MEMBER FUNCTION rational_order -
RETURN DOUBLE PRECISION IS
BEGIN
  RETURN NUMERATOR/DENOMINATOR;
END;
END;
/
```

You can describe the object type RATIONAL as shown in the following example:

```
DESCRIBE rational
```

The following output is displayed:

```
Name                                 Null?      Type
-------------------------------   --------   ------------
NUMERATOR                                      NUMBER
DENOMINATOR                                    NUMBER

METHOD
------
MAP MEMBER FUNCTION RATIONAL_ORDER RETURNS NUMBER
```

The table `PROPERTY` has an `XMLType` column, as shown in the following example:

```
CREATE TABLE PROPERTY (Price NUMBER, Description SYS.XMLTYPE);

Table created.
```

You can describe the table `PROPERTY` as shown in the following example:

```
DESCRIBE property
```

The following output is displayed:

```
Name                                  Null?      Type
------------------------------------  --------   ------------------
PRICE                                            NUMBER
DESCRIPTION                                      SYS.XMLTYPE
```

You can format the output of the `DESCRIBE` command by using the `SET` command, as shown in the following example:

```
SET LINESIZE 80
SET DESCRIBE DEPTH 2
SET DESCRIBE INDENT ON
SET DESCRIBE LINE OFF
```

You can display the settings for an object by using the `SHOW` command, as shown in the following example:

```
SHOW DESCRIBE

DESCRIBE DEPTH 2 LINENUM OFF INDENT ON

DESCRIBE employee
```

The following output is displayed:

```
 Name                                  Null?      Type
 ------------------------------------  --------  ----------------
 FIRST_NAME                                       VARCHAR2(30)
 EMPADDR                                          ADDRESS
 STREET                                           VARCHAR2(20)
 CITY                                             VARCHAR2(20)
```

```
      JOB_ID                                          VARCHAR2(20)
      SALARY                                          NUMBER(7,2)
```

The table ENC_TABLE has an encrypted column COL2, as shown in the following example:

```
CREATE TABLE enc_table (
col1 VARCHAR2(10),
col2 VARCHAR2(15) ENCRYPT,
col3 CHAR(5),
col4 CHAR(20));


Table created.
```

You can describe the table ENC_TABLE as shown in the following example:

```
DESCRIBE enc_table
```

The following output is displayed:

```
Name                                 Null?      Type
------------------------------------ --------   --------------------
COL1                                                VARCHAR2(10)
COL2                                                VARCHAR2(15) ENCRYPT
COL3                                                CHAR(5)
COL4                                                CHAR(20)
```

The table CUSTOMERS has an email domain defined on a table column, as shown in the following example:

```
CREATE DOMAIN Email AS VARCHAR2(30)
DEFAULT ON NULL t_seq.NEXTVAL||'@gmail.com'
CONSTRAINT EMAIL_C CHECK(REGEXP_LIKE (Email, '^(\S+)\@(\S+)\.(\S+)$'))
                     DISPLAY '---' || SUBSTR(Email, INSTR(Email, '@') + 1);
CREATE TABLE customers (Cust_id NUMBER, Cust_email VARCHAR2(100) DOMAIN Email);
```

You can describe the table CUSTOMERS as shown in the following example:

```
DESCRIBE customers
```

The following output is displayed:

```
Name                   Null?   Type
---------------------  -----   ------------------------------------
CUST_ID                        NUMBER
CUST_EMAIL                     VARCHAR2(100) DOMAIN EMAIL
```

When the domain name does not fit into the TYPE column, it wraps automatically to fit in the column.

```
Name                             Null?   Type
-------------------------------  ------- -----------------------------
CUST_ID                                  NUMBER
```

```
CUST_EMAIL                                    VARCHAR2(100) DOMAIN
                                              EMAIL
```

The table `ANNOTATION_TAB` has a column annotation, as shown in the following example:

```
CREATE TABLE annotation_tab
  (c1 NUMBER ANNOTATIONS(EmpGroup2 'Emp_Info', Hidden),
   c2 NUMBER primary key);
```

You can use the `SET DESCRIBE` command to enable displaying the column annotation information:

```
SET DESCRIBE ANNOTATION ON
```

You can describe the table `ANNOTATION_TAB` as shown in the following example:

```
DESCRIBE annotation_tab
```

The following output is displayed:

```
Name          Null?    Type       Annotation
-----------  -------  ---------
-------------------------------------
C1                     NUMBER     EmpGroup2 Emp_Info
                                  Hidden:
C2                     NUMBER
```

For more information on using the `CREATE TYPE` command, see the *Oracle Database SQL Language Reference*.

For information about using the `SET DESCRIBE` and `SHOW DESCRIBE` commands, see the SET and SHOW commands.

13.21 DEL

Syntax

```
DEL [n | n m | n * | n LAST | * | * n | * LAST | LAST]
```

Deletes one or more lines of the buffer.

Terms

Term	Description
n	Deletes line *n*.
n m	Deletes lines *n* through *m*.
n *	Deletes line *n* through the current line.

Term	Description
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

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 DELETE.

Examples

Assume the SQL buffer contains the following query:

```
SELECT LAST_NAME, DEPARTMENT_ID
FROM EMP_DETAILS_VIEW
WHERE JOB_ID = 'SA_MAN'
ORDER BY DEPARTMENT_ID;
```

To make the line containing the WHERE clause the current line, you could enter

```
LIST 3
```

```
3* WHERE JOB_ID = 'SA_MAN'
```

followed by

```
DEL
```

The SQL buffer now contains the following lines:

```
SELECT LAST_NAME, DEPARTMENT_ID
FROM EMP_DETAILS_VIEW
ORDER BY DEPARTMENT_ID
```

To delete the third line of the buffer, enter

```
DEL 3
```

The SQL buffer now contains the following lines:

```
SELECT LAST_NAME, DEPARTMENT_ID
FROM EMP_DETAILS_VIEW
```

13.22 DISCONNECT

Syntax

```
DISC[ONNECT]
```

Commits pending changes to the database and logs the current username out of Oracle Database, but does not exit SQL*Plus.

Usage

Use DISCONNECT within a script to prevent user access to the database when you want to log the user out of Oracle Database but have the user remain in SQL*Plus. In SQL*Plus command-line, use EXIT or QUIT to log out of Oracle Database and return control to your computer's operating system.

Examples

Your script might begin with a CONNECT command and end with a DISCONNECT, as shown later.

```
CONNECT HR
SELECT LAST_NAME, DEPARTMENT_NAME FROM EMP_DETAILS_VIEW;
DISCONNECT
SET INSTANCE FIN2
CONNECT HR2
```

13.23 EDIT

Syntax

```
ED[IT] [file_name[.ext]]
```

where *file_name*[.*ext*] represents the file you wish to edit (typically a script).

Invokes an operating system text editor on the contents of the specified file or on the contents of the buffer.

Enter EDIT with no *filename* to edit the contents of the SQL buffer with the operating system text editor.

Usage

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.

If you specify a *filename*, SQL*Plus searches for the file in the directory set by ORACLE_PATH. If SQL*Plus cannot find the file in ORACLE_PATH, or if

ORACLE_PATH is not set, it searches for the file in the current working directory. If SQL*Plus cannot find the file in either directory, it creates a file with the specified name.

The substitution variable, _EDITOR, contains the name of the text editor invoked by EDIT. You can change the text editor by changing the value of _EDITOR. For information about changing the value of a substitution variable, see DEFINE. EDIT attempts to run the default operating system editor if _EDITOR is undefined.

EDIT places the contents of the SQL buffer in a file named AFIEDT.BUF by default (in your current working directory) and runs 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.

> **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.

In SQL*Plus 9.0 and earlier versions on Windows, the command `EDIT *` opened a blank file after giving an invalid filename warning. In SQL*Plus 10 and later versions on Windows, `EDIT *` gives an invalid filename warning and does not open a blank file. To retain the SQL*Plus 9.0 behavior, enter the set command,

```
SET SQLPLUSCOMPATIBILITY 9.0
```

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.

> **Note:**
>
> In Windows, if you use WordPad as your editor (_EDITOR=write.exe), the buffer is not reloaded when you exit WordPad. In this case, use GET to reload the buffer.

Examples

To edit the file REPORT with the extension SQL using your operating system text editor, enter

```
EDIT REPORT
```

13.24 EXECUTE

Syntax

```
EXEC[UTE] statement
```

where *statement* represents a PL/SQL statement.

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 *Oracle Database PL/SQL Language Reference*.

Usage

If your EXECUTE command cannot fit on one line because of the PL/SQL statement, use the SQL*Plus continuation character (a hyphen).

The length of the command and the PL/SQL statement cannot exceed the length defined by SET LINESIZE.

You can suppress printing of the message "PL/SQL procedure successfully completed" with SET FEEDBACK OFF.

Examples

If the variable :*n* has been defined with:

```
VARIABLE n NUMBER
```

The following EXECUTE command assigns a value to the bind variable *n*:

```
EXECUTE :n := 1


PL/SQL procedure successfully completed.
```

For information on how to create a bind variable, see the VARIABLE command.

13.25 EXIT

Syntax

```
{EXIT | QUIT} [SUCCESS | FAILURE | WARNING | n | variable | :BindVariable]
[COMMIT | ROLLBACK]
```

Commits or rolls back all pending changes, logs out of Oracle Database, terminates SQL*Plus and returns control to the operating system.

Terms

```
{EXIT | QUIT}
```

Can be used interchangeably (QUIT is a synonym for EXIT).

```
SUCCESS
```

Exits normally.

```
FAILURE
```

Exits with a return code indicating failure.

```
WARNING
```

Exits with a return code indicating warning.

```
COMMIT
```

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

EXIT enables you to specify an operating system return code. This enables you to run SQL*Plus scripts in batch mode and to detect programmatically the occurrence of an unexpected event. The manner of detection is operating-system specific.

The key words SUCCESS, WARNING, and FAILURE represent operating-system dependent values. On some systems, WARNING and FAILURE may be indistinguishable.

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.

Examples

The following example commits all uncommitted transactions and returns the error code of the last executed SQL command or PL/SQL block:

```
EXIT SQL.SQLCODE
```

13.26 GET

Syntax

```
GET [FILE] file_name[.ext] [LIST | NOLIST]
```

Loads an operating system file into the SQL buffer.

Terms

`FILE`

Keyword to specify that the following argument is the name of the script you want to load. This optional keyword is usually omitted.

If you want to load a script with the name file, because it is a command keyword, you need to put the name file in single quotes.

`file_name[.ext]`

Represents the file you wish to load (typically a script).

`LIST`

Lists the contents of the file after it is loaded. This is the default.

`NOLIST`

Suppresses the listing.

Usage

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 SET SUF[FIX] {SQL | text}.

If the filename you specify contains the word list or the word file, the name must be 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 SAVE for more information.

Examples

To load a file called YEARENDRPT with the extension SQL into the buffer, enter

`GET YEARENDRPT`

13.27 HELP

Syntax

`HELP |? [topic | <error_code> | <facility name> <error number> ]`

Accesses the SQL*Plus command-line help system, which enables you to get help on SQL*Plus topics. This command also displays error details, such as Cause, Action, and Parameters, for the provided error code, similar to the `OERR` command.

Terms

`topic`

Refers to a SQL*Plus topic, for example, `COLUMN`.

`error_code`

Contains the error prefix, such as `ORA` or `SP2`, and the error number.

`facility name`

Refers to the error prefix, such as `ORA`, `SP2`, and so on.

`error number`

Refers to the error number.

Usage

You can specify only one topic after `HELP`. You can also abbreviate the topic, for example, by using `COL` instead of `COLUMN`. However, if the abbreviation used is ambiguous, then SQL*Plus displays help for all topics that match the abbreviation. For example:

`HELP EX`

SQL*Plus displays the syntax of the `EXECUTE` command, followed by the syntax of the `EXIT` command. If you get a response indicating that help is not available, then consult your database administrator.

You can enter `HELP INDEX` or `? INDEX` for a list of valid topics. Enter `HELP` or `?` without *topic* to get help on the SQL*Plus help system.

> ✎ **See Also:**
>
> Oracle Database Library

When you enter the error number without specifying the facility name in the `HELP` command, then the default facility name `ORA` is used. The message from the `ORA` facility corresponding to the specified error number is displayed.

Examples

You can execute the following command to view a list of SQL*Plus commands:

`HELP INDEX`

or

`? INDEX`

You can execute the following command to view a single column list of SQL*Plus commands:

`HELP TOPICS`

You can execute the following command to view error details for a specified error code:

`HELP ORA-1422`

13.28 HISTORY

Syntax

```
HIST[ORY] [[[N] {R[UN] | E[DIT] | D[ELETE]}] | CLEAR | LIST ]
```

Enables users to run, edit, or delete previously used SQL*Plus, SQL, or PL/SQL commands from the history list in the current session. You can enable or disable the recording of history in the current SQL*Plus session by using the SET HISTORY command.

The HISTORY command enables you to:

- List all entries in the command history list.
- Run an entry in the command history list.
- Edit an entry in the command history list.
- Delete an entry from the command history list.
- Clear all entries in the command history list.

Terms

```
HIST[ORY]
```

Lists all entries in the command history list.

```
N
```

Represents an entry in the command history list. An asterisk (*) indicates the last used command in the command history list. If *N* is omitted, the RUN, EDIT OR DELETE operation is executed in the last used command.

```
R[UN]
```

Enables you to execute entry *N* or the last used command from the command history list.

```
E[DIT]
```

Enables you to edit entry *N* or the last used command in the command history list, using the default text editor. After you edit entry *N* in the command history list and save the changes, a new entry is created at the end of the list. When the number of entries in the command history list reaches the maximum limit, the oldest entry in the list will be cleared to accommodate the new entry.

```
D[ELETE]
```

Enables you to delete entry *N* or the last used command from the command history list. After you delete an entry from the history list, the list is reordered to reflect the most recent changes.

```
CLEAR
```

Enables you to clear all entries in the history list. Once cleared, the history list cannot be recovered.

```
LIST
```

Lists all entries in the history list. This is the same as using the HIST[ORY] command by itself.

Usage

You can use the SQL*Plus DEFINE command to define the variable, _EDITOR, to hold the name of your preferred text editor. For example, to define the editor used by EDIT to be vi, enter the following command:

```
DEFINE _EDITOR = vi
```

EDIT attempts to run the default operating system editor if _EDITOR is undefined. See the DEFINE command for more information.

Example 13-1   Examples

The following example executes the fifth entry in the history list:

```
SQL>history 5 run
```

The following example allows you to edit the third entry in the history list:

```
SQL>history 3 edit
```

The following example allows you to delete the second entry from the history list:

```
SQL>history 2 delete
```

The following example allows you to delete all entries from the history list:

```
SQL>history clear
```

The following example shows you how to edit and run the last used command in the history list:

```
SQL>history edit
SQL>history run
```

The following example shows you how to enable or disable command history, and how to check the command history status:

```
SQL> set history on
SQL> show history
History is ON and set to "100"
SQL> set history off
SQL> show history
History is OFF
SQL> set history 1000
SQL> show history
History is ON and set to "1000"
```

The following example shows you how to list all entries in the history list:

```
SQL> history
  1  show history
  2  show user
```

```
  3  desc dual
* 4  select * from dual;
```

An asterisk (*) indicates the last used command in the command history list.

The following example shows you how to list all entries in the history list, and then execute the second entry:

```
SQL> history
  1  show history
  2  show user
  3  desc dual
* 4  select * from dual;
SQL> history 2 run
USER is "SYSTEM"
SQL> history
  1  show hist
* 2  show user
  3  desc dual
  4  select * from dual;
```

13.29 HOST

Syntax

```
HO[ST] [command]
```

where *command* represents an operating system command.

Executes an operating system command without leaving SQL*Plus.

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 platform-specific Oracle documentation provided for your operating system.

> **Note:**
>
> Operating system commands entered from a SQL*Plus session using the HOST command do not affect the current SQL*Plus session. For example, setting an operating system environment variable only affects SQL*Plus sessions started subsequently.
>
> You can disable HOST. For more information about disabling HOST, see SQL*Plus Security.

Usage

In some operating systems, you can use a character in place of HOST such as "$" in Windows or "!" in UNIX, or you may not have access to the HOST command. See the platform-specific Oracle documentation provided for your operating system or ask your DBA for more information.

On some platforms, an _RC substitution variable may be created with a HOST return value that is operation system dependent. It is recommended that you do not use the _RC substitution variable in scripts as it is not portable.

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 SET SQLT[ERMINATOR] {; | c | ON | OFF} for more information.

Examples

To execute a UNIX operating system command, ls *.sql, enter

```
HOST ls *.sql
```

To execute a Windows operating system command, dir *.sql, enter

```
HOST dir *.sql
```

13.30 INPUT

Syntax

```
I[NPUT] [text]
```

where *text* represents the text you wish to add.

Adds one or more new lines of text after the current line in the buffer.

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 or a period.

Usage

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:

```
SELECT LAST_NAME, DEPARTMENT_ID, SALARY, COMMISSION_PCT
FROM EMP_DETAILS_VIEW
```

To add an ORDER BY clause to the query, enter

```
LIST 2
```

```
2* FROM EMP_DETAILS_VIEW
```

```
INPUT ORDER BY LAST_NAME
```

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 LAST_NAME, DEPARTMENT_ID, SALARY, COMMISSION_PCT
2 FROM EMP_DETAILS_VIEW
3* ORDER BY LAST_NAME
```

To add a two-line WHERE clause, enter

```
LIST 2


2* FROM EMP_DETAILS_VIEW


INPUT
  3  WHERE JOB_ID = 'SA_MAN'
  4  AND COMMISSION_PCT=.25
  5
```

INPUT prompts you for new lines until you enter an empty line or a period. The SQL buffer now contains the following lines:

```
SELECT LAST_NAME, DEPARTMENT_ID, SALARY, COMMISSION_PCT
FROM EMP_DETAILS_VIEW
WHERE JOB_ID = 'SA_MAN'
AND COMMISSION_PCT = .25
ORDER BY LAST_NAME
```

13.31 LIST

Syntax

```
L[IST] [n | n m | n * | n LAST | * | * n | * LAST | LAST]
```

Lists one or more lines of the SQL buffer.

In SQL*Plus command-line you can also use ";" to list all the lines in the SQL buffer.

Terms

Term	Description
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.

Term	Description
* *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, or ";" to list all lines. The last line listed becomes the new current line (marked by an asterisk).

Examples

To list the contents of the buffer, enter

```
LIST
```

or enter

```
;
```

```
1 SELECT LAST_NAME, DEPARTMENT_ID, JOB_ID
2 FROM EMP_DETAILS_VIEW
3 WHERE JOB_ID = 'SH_CLERK'
4* ORDER BY DEPARTMENT_ID
```

The asterisk indicates that line 4 is the current line.

To list the second line only, enter

```
LIST 2
```

The second line is displayed:

```
2* FROM EMP_DETAILS_VIEW
```

To list from the current line (now line 2) to the last line, enter

```
LIST * LAST
```

You will then see this:

```
2 FROM EMP_DETAILS_VIEW
3 WHERE JOB_ID = 'SH_CLERK'
4* ORDER BY DEPARTMENT_ID
```

13.32 OERR

Syntax

```
OERR [error_code|<facility name> <error number>]
```

Displays detailed cause and action text for Oracle errors. The `OERR` command accepts an error code or an error prefix (facility name) along with an error number to display error details.

Terms

`error_code`

Contains the error prefix, such as `ORA` or `SP2`, and the error number.

`facility name`

Refers to the error prefix, such as `ORA`, `SP2`, and so on.

`error number`

Refers to the error number.

Examples

The following examples display the usage and output of the `OERR` command:

Example 1

```
OERR ORA-1033
Message: "ORACLE initialization or shutdown in progress"
Help: https://docs.oracle.com/error-help/db/ora-01033/
Cause:  An attempt was made to log on while Oracle is being started up
        or shutdown.
Action: Wait a few minutes. Then retry the operation.
```

Example 2

```
OERR ORA 1033
Message: "ORACLE initialization or shutdown in progress"
Help: https://docs.oracle.com/error-help/db/ora-01033/
Cause:  An attempt was made to log on while Oracle is being started up
        or shutdown.
Action: Wait a few minutes. Then retry the operation.
```

Example 3

```
OERR SP2 0640
Message: "Not connected\n"
Help: https://docs.oracle.com/error-help/db/sp2-00640/
Cause:  The PASSWORD command was issued when there was no
        connection to the Oracle instance.
Action: Connect to the Oracle database before re-issuing the
        PASSWORD command.
```

13.33 PASSWORD

Syntax

```
PASSW[ORD] [username]
```

where *username* specifies the user. If omitted, *username* defaults to the current user.

Enables you to change a password without echoing it on an input device.

If the user's password profile contains a non-zero value for the `PASSWORD_ROLLOVER_TIME` parameter, then changing the password starts the password rollover period for the user account.

If you are a database user that is associated with a profile that enables gradual password rollover (a period of time where both the old and and new passwords are valid), then you can change your password through SQL*Plus and have both the newly created password and the old password work during user authentication.

For more information, see Gradual Database Password Rollover for Applications in *Oracle Database Security Guide*.

During the password rollover period, you can log in using either the old password (*password1*) or the new password (*password2*).

To change the password again during the rollover period, use either *password1* or *password2* as the old password. For example, use either one of the following statements to change the password to *password3*:

```
ALTER USER jones IDENTIFIED BY password3 REPLACE password1
```

```
ALTER USER jones IDENTIFIED BY password3 REPLACE password2
```

There is no limit to the number of times you can change the password during the password rollover period. However, note that the password rollover period is fixed starting from the time the password is changed the first time.

Usage

To change the password of another user, you must have been granted the appropriate privilege. See CONNECT for more information about changing your password.

Examples

If you want to change your current password, enter

```
PASSWORD
Changing password for your_password
Old password: your_password
New password: new_password
Retype new password: new_password
Password changed
```

If you are logged on as a DBA, and want to change the password for user johnw (currently identified by johnwpass) to johnwnewpass

```
PASSWORD johnw
Changing password for johnw
New password: johnwnewpass
Retype new password: johnwnewpass
Password changed
```

Passwords are not echoed to the screen, they are shown here for your convenience.

13.34 PAUSE

Syntax

```
PAU[SE] [text]
```

where *text* represents the text you wish to display.

Displays the specified text then waits for the user to press RETURN.

Enter PAUSE followed by no text to display two empty lines.

Usage

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.

See SET PAU[SE] {ON | OFF | text} for information on pausing between pages of a report.

Examples

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 script:

```
SET PAUSE OFF
PAUSE Adjust paper and press RETURN to continue.
SELECT ...
```

13.35 PING

Syntax

```
PING [[LISTENER] connect_identifier]
```

Pings the database or database network listener to check availability. The `PING` command will fail with an error if either the database or the network listener are unavailable. It indicates that either the network listener or the database needs to be restarted, or that the database connection is not usable and needs to be recreated.

When the `PING` command is executed without a connect identifier, then a round-trip message is sent through the network listener to the currently connected database. When it is executed with a connect identifier, then a round-trip message is sent to the specified network listener to verify if it is able to handle the database connections.

Terms

`LISTENER`

Determines whether the target listener specified in the connect identifier is reachable and displays the total time taken for a round trip to ping the listener. This is similar to Oracle Database's TNSPING utility. The output also displays the location of the `tnsnames.ora` file, if it is used.

`connect_identifier`

Refers to an Oracle Net connect identifier.

Examples

In the following example, the `PING` command is executed to ping the database of the current connection:

```
PING
Ok (2.669 msec)
```

In the following example, the `PING` command is executed to ping the network listener with the specified alias:

```
PING INST1
Network service name mapping file: /opt/oracle/tnsnames.ora
Attempting to contact: (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=cdb1_pdb1))
(ADDRESS=(PROTOCOL=tcp)(HOST=host1.customer.domain.com)(PORT=1521)))
Ok (1.588 msec)
```

In the following example, the `PING` command fails with an error message:

```
PING INST2
Local Net Naming configuration file: /opt/oracle/tnsnames.ora
Attempting to contact: (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=cdb1_pdb1))
(ADDRESS=(PROTOCOL=tcp)(HOST=host1.customer.domain.com)(PORT=1521)))
SP2-1683: Ping failed with error TNS-12560.
```

13.36 PRINT

Syntax

```
PRINT [variable ...]
```

where *variable* ... represents names of bind variables whose values you want to display.

Displays the current values of bind variables.

Enter PRINT with no variables to print all bind variables.

Usage

Bind variables are created using the VARIABLE command. See VARIABLE for more information and examples.

You can control the formatting of the PRINT output just as you would query output. For more information, see the formatting techniques described in Formatting SQL*Plus Reports.

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. See SET for more information.

Examples

The following example illustrates a PRINT command:

```
VARIABLE n NUMBER
BEGIN
:n := 1;
END;
/

PL/SQL procedure successfully completed.

PRINT n

N
----------
1
```

13.37 PROMPT

Syntax

```
PRO[MPT] [text]
```

where *text* represents the text of the message you want to display.

Sends the specified message or a blank line to the user's screen. If you omit *text*, PROMPT displays a blank line on the user's screen.

Usage

You can use this command in scripts to give information to the user.

Examples

The following example shows the use of PROMPT in conjunction with ACCEPT in a script called ASKFORDEPT.SQL. ASKFORDEPT.SQL contains the following SQL*Plus and SQL commands:

```
PROMPTPROMPT Please enter a valid departmentPROMPT For example:  10SELECT
DEPARTMENT_NAME FROM EMP_DETAILS_VIEWWHERE DEPARTMENT_ID = &NEWDEPT
```

Assume you run the file using START or @:

```
@ASKFORDEPT.SQL VAL1
@HTTP://machine_name.domain:port/ASKFORDEPT.SQL VAL1


Please enter a valid department
For example: 10
Department ID?>
```

You can enter a department number at the prompt Department ID?>. 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 Department ID?> prompt. You can use SET VERIFY OFF to prevent this behavior.

13.38 RECOVER

Syntax

```
RECOVER {general | managed | BEGIN BACKUP | END BACKUP}
```

where the *general* clause has the following syntax:

```
[AUTOMATIC] [FROM location]
{ {full_database_recovery  | partial_database_recovery  | LOGFILE filename}
[ {TEST | ALLOW integer CORRUPTION | parallel_clause }  [TEST
| ALLOW integer CORRUPTION | parallel_clause  ]...]| CONTINUE [DEFAULT] | CANCEL}
```

where the *full_database_recovery* clause has the following syntax:

```
[STANDBY] DATABASE
 [ {UNTIL {CANCEL | TIME date | CHANGE integer} | USING BACKUP CONTROLFILE
 | SNAPSHOT TIME date}...]
```

where the *partial_database_recovery* clause has the following syntax:

```
{TABLESPACE tablespace [, tablespace]...
  | DATAFILE {filename | filenumber} [, filename | filenumber]...
  | STANDBY {TABLESPACE tablespace [, tablespace]...
  | DATAFILE {filename | filenumber} [, filename | filenumber]...}
    UNTIL [CONSISTENT WITH] CONTROLFILE }
```

where the *parallel* clause has the following syntax:

```
{ NOPARALLEL | PARALLEL [ integer ] }
```

where the *managed* clause has the following syntax:

```
MANAGED STANDBY DATABASE recover_clause | cancel_clause | finish_clause
```

where the *recover_clause* has the following syntax:

```
{ { DISCONNECT [ FROM SESSION ]  | { TIMEOUT integer | NOTIMEOUT }  }
  | { NODELAY | DEFAULT DELAY | DELAY integer }
  | NEXT integer  | { EXPIRE integer | NO EXPIRE }
  | parallel_clause | USING CURRENT LOGFILE | UNTIL CHANGE integer
  | THROUGH { [ THREAD integer ] SEQUENCE integer
  | ALL ARCHIVELOG  | { ALL | LAST | NEXT } SWITCHOVER}} ...
```

where the *cancel_clause* has the following syntax:

```
CANCEL [IMMEDIATE] [WAIT | NOWAIT]
```

where the *finish_clause* has the following syntax:

```
[ DISCONNECT [ FROM SESSION ] ] [ parallel_clause ]
FINISH [ SKIP [ STANDBY LOGFILE ] ] [ WAIT | NOWAIT ]
```

where the *parallel_clause* has the following syntax:

```
{ NOPARALLEL | PARALLEL [ integer ] }
```

Performs media recovery on one or more tablespaces, one or more datafiles, or the entire database. For more information on the RECOVER command, see the *Oracle Database Administrator's Guide*, the ALTER DATABASE RECOVER command in the *Oracle Database SQL Language Reference*, and the *Oracle Database Backup and Recovery User's Guide* guide.

Terms

```
AUTOMATIC
```

Automatically generates the name of the next archived redo log file needed to continue the recovery operation. Oracle Database uses the LOG_ARCHIVE_DEST (or LOG_ARCHIVE_DEST_ 1) and LOG_ARCHIVE_FORMAT parameters (or their defaults) to generate the target redo log filename. If the file is found, the redo contained in that file is applied. If the file is not found, SQL*Plus prompts you for a filename, displaying a generated filename as a suggestion.

If you do not specify either AUTOMATIC or LOGFILE, SQL*Plus prompts you for a filename, suggesting the generated filename. You can either accept the generated filename or replace it with a fully qualified filename. You can save time by using the LOGFILE clause to specify the filename if you know the archived filename differs from the filename Oracle Database would generate.

```
FROM location
```

Specifies the location from which the archived redo log file group is read. The value of *location* must be a fully specified file location. If you omit this parameter, SQL*Plus assumes the archived redo log file group is in the location specified by the initialization parameter LOG_ARCHIVE_DEST or LOG_ARCHIVE_DEST_1. Do not specify FROM if you have set a file with SET LOGSOURCE.

```
full_database_recovery
```

Enables you to specify the recovery of a full database.

```
partial_database_recovery
```

Enables you to specify the recovery of individual tablespaces and datafiles.

```
LOGFILE
```

Continues media recovery by applying the specified redo log file. In interactive recovery mode (AUTORECOVERY OFF), if a bad log name is entered, errors for the bad log name are displayed and you are prompted to enter a new log name.

```
TEST
```

Specifies a trial recovery to detect possible problems. Redo is applied normally, but no changes are written to disk, and changes are rolled back at the end of the trial recovery. You can only use the TEST clause for a trial recovery if you have restored a backup. In the event of logfile corruption, specifies the number of corrupt blocks that

can be tolerated while allowing recovery to proceed. During normal recovery, *integer* cannot exceed 1.

```
ALLOW integer CORRUPTION
```

In the event of logfile corruption, specifies the number of corrupt blocks that can be tolerated while allowing recovery to proceed.

```
parallel _clause
```

Enables you to specify the degree of parallel processing to use during the recovery operation.

```
CONTINUE
```

Continues multi-instance recovery after it has been interrupted to disable a thread.

```
CONTINUE DEFAULT
```

Continues recovery using the redo log file generated automatically by Oracle Database if no other logfile is specified. This is equivalent to specifying AUTOMATIC, except that Oracle Database does not prompt for a filename.

```
CANCEL
```

Terminates cancel-based recovery.

```
SNAPSHOT TIME date
```

Recovers the database with a storage snapshot using Storage snapshot Optimization.

```
STANDBY DATABASE
```

Recovers the standby database using the control file and archived redo log files copied from the primary database. The standby database must be mounted but not open.

```
DATABASE
```

Recovers the entire database.

```
UNTIL CANCEL
```

Specifies an incomplete, cancel-based recovery. Recovery proceeds by prompting you with suggested filenames of archived redo log files, and recovery completes when you specify CANCEL instead of a filename.

```
UNTIL TIME
```

Specifies an incomplete, time-based recovery. Use single quotes, and the following format:

```
'YYYY-MM-DD:HH24:MI:SS'
```

```
UNTIL CHANGE
```

Specifies an incomplete, change-based recovery. *integer* is the number of the System Change Number (SCN) following the last change you wish to recover. For example, if you want to restore your database up to the transaction with an SCN of 9, you would specify UNTIL CHANGE 10.

```
USING BACKUP CONTROLFILE
```

Specifies that a backup of the control file be used instead of the current control file.

```
TABLESPACE
```

Recovers a particular tablespace. *tablespace* is the name of a tablespace in the current database. You may recover up to 16 tablespaces in one statement.

```
DATAFILE
```

Recovers a particular datafile. You can specify any number of datafiles.

```
STANDBY TABLESPACE
```

Reconstructs a lost or damaged tablespace in the standby database using archived redo log files copied from the primary database and a control file.

```
STANDBY DATAFILE
```

Reconstructs a lost or damaged datafile in the standby database using archived redo log files copied from the primary database and a control file.

```
UNTIL CONSISTENT WITH CONTROLFILE
```

Specifies that the recovery of an old standby datafile or tablespace uses the current standby database control file.

```
PARALLEL [integer]
```

This is the default. SQL*Plus selects a degree of parallelism equal to the number of CPUs available on all participating instances times the value of the PARALLEL_THREADS_PER_CPU initialization parameter.

The PARALLEL keyword overrides the RECOVERY_PARALLELISM initialization parameter. For more information about the PARALLEL keyword see the *Oracle Real Application Clusters Administration and Deployment Guide* guide.

Use *integer* to specify the degree of parallelism, which is the number of parallel threads used in the parallel operation. Each parallel thread may use one or two parallel execution processes.

```
NOPARALLEL
```

Specifies serial recovery processing.

```
MANAGED STANDBY DATABASE
```

Specifies sustained standby recovery mode. This mode assumes that the standby database is an active component of an overall standby database architecture. A primary database actively archives its redo log files to the standby site. As these archived redo logs arrive at the standby site, they become available for use by a managed standby recovery operation. Sustained standby recovery is restricted to media recovery.

For more information on the parameters of this clause, see the *Oracle Database Backup and Recovery User's Guide*.

```
DISCONNECT
```

Indicates that the managed redo process (MRP) should apply archived redo files as a detached background process. Doing so leaves the current session available.

```
TIMEOUT
```

Specifies in minutes the wait period of the sustained recovery operation. The recovery process waits for *integer* minutes for a requested archived log redo to be available for writing to the standby database. If the redo log file does not become available within that time, the recovery process terminates with an error message. You can then issue the statement again to return to sustained standby recovery mode.

If you do not specify this clause, or if you specify NOTIMEOUT, the database remains in sustained standby recovery mode until you reissue the statement with the RECOVER CANCEL clause or until instance shutdown or failure.

```
NODELAY
```

Applies a delayed archivelog immediately to the standby database overriding any DELAY setting in the LOG_ARCHIVE_DEST_n parameter on the primary database. If you omit this clause, application of the archivelog is delayed according to the parameter setting. If DELAY was not specified in the parameter, the archivelog is applied immediately.

```
DEFAULT DELAY
```

Waits the default number of minutes specified in the LOG_ARCHIVE_DEST_n initialization parameter before applying the archived redo logs.

```
DELAY integer
```

Waits *integer* minutes before applying the archived redo logs.

```
NEXT integer
```

Applies the specified number of archived redo logs as soon as possible after they have been archived. It temporarily overrides any DELAY setting in the LOG_ARCHIVE_DEST_n parameter on the primary database, and any delay values set in an earlier SQL*Plus RECOVER command or an ALTER DATABASE RECOVER command.

```
EXPIRE integer
```

Specifies the number of minutes from the current time after which managed recovery terminates automatically.

```
NO EXPIRE
```

Disables a previously specified EXPIRE *integer* option.

```
USING CURRENT LOGFILE
```

Recovers redo from standby online logs as they are being filled, without requiring them to be archived in the standby database first.

```
UNTIL CHANGE integer
```

Processes managed recovery up to but not including the specified system change number (SCN).

```
THROUGH THREAD integer SEQUENCE integer
```

Terminates managed recovery based on archivelog thread number and sequence number. Managed recovery terminates when the corresponding archivelog has been applied. If omitted, THREAD defaults to 1.

```
THROUGH ALL ARCHIVELOG
```

Continues managed standby until all archivelogs have been recovered. You can use this statement to override a THROUGH THREAD *integer* SEQUENCE *integer* clause issued in an earlier statement. If the THROUGH clause is omitted, this is the default.

```
THROUGH ALL SWITCHOVER
```

Keeps managed standby recovery running through all switchover operations.

```
THROUGH LAST  SWITCHOVER
```

Terminates managed standby recovery after the final end-of-redo archival indicator.

```
THROUGH NEXT  SWITCHOVER
```

Terminates managed standby recovery after recovering the next end-of-redo archival indicator.

```
CANCEL (managed clause)
```

Terminates managed standby recovery after applying the current archived redo file. Session control returns when the recovery process terminates.

```
CANCEL IMMEDIATE
```

Terminates managed standby recovery after applying the current archived redo file, or after the next redo log file read, whichever comes first. Session control returns when the recovery process terminates.

```
CANCEL IMMEDIATE WAIT
```

Terminates managed standby recovery after applying the current archived redo file or after the next redo log file read, whichever comes first. Session control returns when the managed standby recovery terminates.

CANCEL IMMEDIATE cannot be issued from the same session that issued the RECOVER MANAGED STANDBY DATABASE statement.

```
CANCEL IMMEDIATE NOWAIT
```

Terminates managed standby recovery after applying the current archived redo file, or after the next redo log file read, whichever comes first. Session control returns immediately.

```
CANCEL NOWAIT
```

Terminates managed standby recovery after the next redo log file read and returns session control immediately.

```
FINISH
```

Recovers the current standby online logfiles of the standby database. This clause may be useful if the primary database fails. It overrides any delays specified for archivelogs, so that logs are applied immediately.

FINISH cannot be issued if you have also specified TIMEOUT, DELAY, EXPIRE or NEXT clauses.

Usage

You must have the OSDBA role enabled. You cannot use the RECOVER command when connected through the multi-threaded server.

To perform media recovery on an entire database (all tablespaces), the database must be mounted and closed, and all tablespaces requiring recovery must be online.

To perform media recovery on a tablespace, the database must be mounted or open, and the tablespace must be offline.

To perform media recovery on a datafile, the database can remain open and mounted with the damaged datafiles offline (unless the file is part of the SYSTEM tablespace).

Before using the RECOVER command you must have restored copies of the damaged datafiles from a previous backup. Be sure you can access all archived and online redo log files dating back to when that backup was made.

When another log file is required during recovery, a prompt suggests the names of files that are needed. The name is derived from the values specified in the initialization parameters LOG_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT. You should restore copies of the archived redo log files needed for recovery to the destination specified in LOG_ARCHIVE_DEST, if necessary. You can override the initialization parameters by setting the LOGSOURCE variable with the SET LOGSOURCE command.

During recovery you can accept the suggested log name by pressing return, cancel recovery by entering CANCEL instead of a log name, or enter AUTO at the prompt for automatic file selection without further prompting.

If you have enabled autorecovery (that is, SET AUTORECOVERY ON), recovery proceeds without prompting you with filenames. Status messages are displayed when each log file is applied. When normal media recovery is done, a completion status is returned.

Examples

To recover the entire database, enter

```
RECOVER DATABASE
```

To recover the database until a specified time, enter

```
RECOVER DATABASE UNTIL TIME 01-JAN-2001:04:32:00
```

To recover the two tablespaces ts_one and ts_two from the database, enter

```
RECOVER TABLESPACE ts_one, ts_two
```

To recover the datafile data1.db from the database, enter

```
RECOVER DATAFILE 'data1.db'
```

13.39 REMARK

Syntax

```
REM[ARK]
```

Begins a comment in a script. SQL*Plus does not interpret the comment as a command.

Usage

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.

A "–" at the end of a REMARK line is treated as a line continuation character.

For details on entering comments in scripts using the SQL comment delimiters, /* ... */, or the ANSI/ISO comment delimiter, - -, see About Placing Comments in Scripts.

Examples

The following script 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_ID" ON REPORT
REM Each column displays the sums of salaries by job for
REM one of the departments 10, 20, 30.
SELECT JOB_ID,
SUM(DECODE( DEPARTMENT_ID, 10, SALARY, 0)) "DEPARTMENT 10",
SUM(DECODE( DEPARTMENT_ID, 20, SALARY, 0)) "DEPARTMENT 20",
SUM(DECODE( DEPARTMENT_ID, 30, SALARY, 0)) "DEPARTMENT 30",
SUM(SALARY) "TOTAL BY JOB_ID"
FROM EMP_DETAILS_VIEW
GROUP BY JOB_ID;
```

13.40 REPFOOTER

Syntax

```
REPF[OOTER] [PAGE] [printspec [text | variable] ...] | [ON | OFF]
```

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
```

Places and formats a specified report footer at the bottom of each report, or lists the current REPFOOTER definition.

Enter REPFOOTER with no clauses to list the current REPFOOTER definition.

Terms

See the REPHEADER command for additional information on terms and clauses in the REPFOOTER command syntax.

Usage

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.

Examples

To define "END EMPLOYEE LISTING REPORT" as a report footer on a separate page and to center it, enter:

```
REPFOOTER PAGE CENTER 'END EMPLOYEE LISTING REPORT'
TTITLE RIGHT 'Page: ' FORMAT 999 SQL.PNO
SELECT LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY > 12000;


LAST_NAME                    SALARY
------------------------ ----------
King                          24000
Kochhar                       17000
De Haan                       17000
Russell                       14000
Partners                      13500
Hartstein                     13000
                         ----------
sum                           98500


                                                              Page:
2
                          END EMPLOYEE LISTING REPORT

6 rows selected.
```

To suppress the report footer without changing its definition, enter

```
REPFOOTER OFF
```

13.41 REPHEADER

Syntax

```
REPH[EADER] [PAGE] [printspec [text | variable] ...]  | [ON | OFF]
```

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
```

Places and formats a specified report header at the top of each report, or lists the current REPHEADER definition.

Enter REPHEADER with no clauses to list the current REPHEADER definition.

Terms

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.

```
text
```

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
```

A substitution variable or any of the following system-maintained values. SQL.LNO is the current line number, SQL.PNO is the current page number, SQL.CODE is the current error code, SQL.RELEASE is the current Oracle Database release number, and SQL.USER is the current username.

To print one of these values, reference the appropriate variable in the report header or footer. You can use the FORMAT clause to format *variable*.

```
OFF
```

Turns the report header or footer off (suppresses its display) without affecting its definition.

```
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] 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 bold text on three consecutive lines, instead of bold.

```
FORMAT text
```

Specifies a format model that determines the format of 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 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 using the default format.

Usage

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, positioning and formatting each constant or variable as specified by the *printspec* clauses that precede it.

Examples

To define "EMPLOYEE LISTING REPORT" as a report header on a separate page, and to center it, enter:

```
REPHEADER PAGE CENTER 'EMPLOYEE LISTING REPORT'
TTITLE RIGHT 'Page: ' FORMAT 999 SQL.PNO
SELECT LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY > 12000;
```

```
                                                             Page:    1
                          EMPLOYEE LISTING REPORT

                                                             Page:    2
LAST_NAME                      SALARY
------------------------- ----------
King                            24000
Kochhar                         17000
De Haan                         17000
Russell                         14000
Partners                        13500
Hartstein                       13000
                           ----------
sum                             98500

6 rows selected.
```

To suppress the report header without changing its definition, enter:

```
REPHEADER OFF
```

13.42 RUN

Syntax

```
R[UN]
```

Lists and executes the SQL command or PL/SQL block currently stored in the SQL buffer.

Usage

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. The SQL buffer always contains the last SQL statement or PL/SQL block entered.

Examples

Assume the SQL buffer contains the following script:

```
SELECT DEPARTMENT_ID
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000
```

To RUN the script, enter

```
RUN


  1  SELECT DEPARTMENT_ID
  2   FROM EMP_DETAILS_VIEW
  3 WHERE SALARY>12000


DEPARTMENT_ID
-------------
           90
           90
           90
           80
           80
           20

6 rows selected.
```

13.43 SAVE

Syntax

```
SAV[E] [FILE] file_name[.ext] [CRE[ATE] | REP[LACE] | APP[END]]
```

Saves the contents of the SQL buffer in an operating system script.

Terms

```
FILE
```

Keyword to specify that the following argument is the name you want to give to the saved script. This optional keyword is usually omitted.

If you want to save the script with the name file, because it is a command keyword, you need to put the name file in single quotes.

```
file_name[.ext]
```

Specifies the script in which you wish to save the buffer's contents.

```
CREATE
```

Creates a new file with the name specified. This is the default behavior.

```
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

If you do not specify an extension, SQL*Plus assumes the default command-file extension (normally SQL). See SET SUF[FIX] {SQL | text} for information on changing this default extension.

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.

Examples

To save the contents of the buffer in a file named DEPTSALRPT with the extension SQL, enter

```
SAVE DEPTSALRPT
```

To save the contents of the buffer in a file named DEPTSALRPT with the extension OLD, enter

```
SAVE DEPTSALRPT.OLD
```

13.44 SET

Sets a system variable to alter the SQL*Plus environment settings for your current session, for example, to:

- customize HTML formatting
- enable or disable the printing of column headings
- set the number of lines per page
- set the display width for data

Syntax

```
SET system_variable value
```

where *system_variable* and *value* represent one of the clauses shown in the SET System Variable Summary table following.

Usage

SQL*Plus maintains system variables (also called SET command variables) to enable you to set up 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. The default value for each system variable is underlined in the following sections.

SET ROLE and SET TRANSACTION are SQL commands (see SQL Statements: MERGE to UPDATE for more information). When not followed by the keywords TRANSACTION or ROLE, SET is assumed to be a SQL*Plus command.

13.45 SET System Variable Summary

System Variable	Description
SET APPI[NFO]{ON | OFF | text}	Sets automatic registering of scripts through the DBMS_APPLICATION_INFO package.
SET ARRAY[SIZE] {15 | n}	Sets the number of rows, called a *batch*, that SQL*Plus will fetch from the database at one time.
SET AUTO[COMMIT]{ON | OFF | IMM[EDIATE] | n}	Controls when Oracle Database commits pending changes to the database.
SET AUTOP[RINT] {ON | OFF}	Sets the automatic printing of bind variables.
SET AUTORECOVERY [ON | OFF]	ON sets the RECOVER command to automatically apply the default filenames of archived redo log files needed during recovery.
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]	Displays a report on the execution of successful SQL DML statements (SELECT, INSERT, UPDATE, DELETE or MERGE).
SET BLO[CKTERMINATOR] {. | c | ON | OFF}	Sets the non-alphanumeric character used to end PL/SQL blocks to *c*.
SET CMDS[EP] {; | c | ON | OFF}	Sets the non-alphanumeric character used to separate multiple SQL*Plus commands entered on one line to *c*.
SET COLINVI[SIBLE] [ON | OFF]	ON sets the DESCRIBE command to display column information for an invisible column..
SET COLSEP { | text}	Sets the text to be printed between selected columns.
SET CON[CAT] {. | c | ON | OFF}	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.
SET COPYC[OMMIT] {0 | n}	Controls the number of batches after which the COPY command commits changes to the database.
SET COPYTYPECHECK {ON | OFF}	Sets the suppression of the comparison of datatypes while inserting or appending to tables with the COPY command.
SET DEF[INE] {& | c | ON | OFF}	Sets the character used to prefix variables to *c*.
SET DESCRIBE [DEPTH {1 | n | ALL}] [LINENUM {ON | OFF}] [INDENT {ON | OFF}] [[ANNO]TATION {ON|OFF}]	Sets the depth of the level to which you can recursively describe an object.
SET ECHO {ON | OFF}	Controls whether the START command lists each command in a script as the command is executed.

System Variable	Description
SET EDITF[ILE] file_name[.ext]	Sets the default filename for the EDIT command.
**SET EMB[EDDED] {ON	OFF}**	Controls where on a page each report begins.
**SET ERRORDETAILS { OFF	ON	VERBOSE }**	Displays the Oracle Database Error Help URL along with the error message cause and action details when any SQL, PL/SQL, or SQL*Plus statement fails during execution.
**SET ERRORL[OGGING] {ON	OFF} [TABLE [schema.]tablename] [TRUNCATE] [IDENTIFIER identifier]**	Enables recording of SQL, PL/SQL and SQL*Plus errors to an error log table which you can query later.
**SET ESC[APE] {\	c	ON	OFF}**	Defines the character you enter as the escape character.
**SET ESCCHAR {@	?	%	OFF}**	Specifies a special character to escape in a filename. Prevents character translation causing an error.
**SET EXITC[OMMIT] {ON	OFF}**	Specifies whether the default EXIT behavior is COMMIT or ROLLBACK.
**SET FEED[BACK] {6	n	ON	OFF	ONLY}] [SQL_ID]**	Displays the number of records returned by a query when a query selects at least *n* records.
**SET FLAGGER {OFF	ENTRY	INTERMED[IATE]	FULL}**	Checks to make sure that SQL statements conform to the ANSI/ISO SQL92 standard.
**SET FLU[SH] {ON	OFF}**	Controls when output is sent to the user's display device.
**SET HEA[DING] {ON	OFF}**	Controls printing of column headings in reports.
**SET HEADS[EP] {	c	ON	OFF}**	Defines the character you enter as the heading separator character.
**SET HIST[ORY] {ON	OFF	n}**	Enables or disables the history of commands and SQL or PL/SQL statements issued in the current SQL*Plus session.
**SET INSTANCE [instance_path	LOCAL]**	Changes the default instance for your session to the specified instance path.
SET JSONPRINT	Formats the output of JSON type columns.
**SET LIN[ESIZE] {80	n	WINDOW}**	Sets the total number of characters that SQL*Plus displays on one line before beginning a new line.
**SET LOBOF[FSET] {1	n}**	Sets the starting position from which BLOB, BFILE, CLOB and NCLOB data is retrieved and displayed.
**SET LOBPREFETCH {0	n}**	Sets the amount of LOB data that SQL*Plus will prefetch from the database at one time.
SET LOGSOURCE [pathname]	Specifies the location from which archive logs are retrieved during recovery.

System Variable	Description
SET LONG {80 | n}	Sets maximum width (in bytes) for displaying LONG, BLOB, BFILE, CLOB, NCLOB and XMLType values; and for copying LONG values.
SET LONGC[HUNKSIZE] {80 | n}	Sets the size (in bytes) of the increments in which SQL*Plus retrieves a LONG, BLOB, BFILE, CLOB, NCLOB or XMLType value.
SET MARK[UP]	Outputs CSV format data or HTML marked up text.
SET NEWP[AGE] {1 | n | NONE}	Sets the number of blank lines to be printed from the top of each page to the top title.
SET NULL text	Sets the text that represents a null value in the result of a SQL SELECT command.
SET NUMF[ORMAT] format	Sets the default format for displaying numbers.
SET NUM[WIDTH] {10 | n}	Sets the default width for displaying numbers.
SET PAGES[IZE] {14 | n}	Sets the number of lines in each page.
SET PAU[SE] {ON | OFF | text}	Enables you to control scrolling of your terminal when running reports.
SET RECSEP {WR[APPED] | EA[CH] | OFF}	RECSEP tells SQL*Plus where to make the record separation.
SET RECSEPCHAR { | c}	Display or print record separators.
SET ROWLIMIT {n | OFF}	Sets a limit for the number of rows to display for a query.
SET ROWPREFETCH {15 | n}	Sets the number of rows that SQL*Plus will prefetch from the database at one time.
SET SECUREDCOL {OFF | ON} [UNAUTH[ORIZED] text] [UNK[NOWN] text]	Sets how secure column values are displayed for users without permission to view a column and for columns with unknown security.
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNL[IMITED]}] [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.
SET SHIFT[INOUT] {VIS[IBLE] | INV[ISIBLE]}	Enables correct alignment for terminals that display shift characters.
SET SHOW[MODE] {ON | OFF}	Controls whether SQL*Plus lists the old and new settings of a SQL*Plus system variable when you change the setting with SET.
SET SQLBL[ANKLINES] {ON | OFF}	Controls whether SQL*Plus puts blank lines within a SQL command or script.
SET SQLC[ASE] {MIX[ED] | LO[WER] | UP[PER]}	Converts the case of SQL commands and PL/SQL blocks just prior to execution.

System Variable	Description
SET 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 (–).
SET SQLN[UMBER] {ON | OFF}	Sets the prompt for the second and subsequent lines of a SQL command or PL/SQL block.
SET SQLPLUSCOMPAT[IBILITY] {x.y[.z]}	Sets the behavior or output format of VARIABLE to that of the release or version specified by *x.y*[.*z*].
SET SQLPRE[FIX] {# | c}	Sets the SQL*Plus prefix character.
SET SQLP[ROMPT] {SQL> | text}	Sets the SQL*Plus command prompt.
SET SQLT[ERMINATOR] {; | c | ON | OFF}	Sets the character used to end and execute SQL commands to *c*.
SET STATEMENTC[ACHE] {0 | n}	Sets the statement cache size .
SET SUF[FIX] {SQL | text}	Sets the default file that SQL*Plus uses in commands that refer to scripts.
SET TAB {ON | OFF}	Determines how SQL*Plus formats white space in terminal output.
SET TERM[OUT] {ON | OFF}	Controls the display of output generated by commands executed from a script.
SET TI[ME] {ON | OFF}	Controls the display of the current time.
SET TIMI[NG] {ON | OFF}	Controls the display of timing statistics.
SET TRIM[OUT] {ON | OFF}	Determines whether SQL*Plus puts trailing blanks at the end of each displayed line.
SET TRIMS[POOL] {ON | OFF}	Determines whether SQL*Plus puts trailing blanks at the end of each spooled line.
SET UND[ERLINE] {- | c | ON | OFF}	Sets the character used to underline column headings in SQL*Plus reports to *c*.
SET VER[IFY] {ON | OFF}	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.
SET WRA[P] {ON | OFF}	Controls whether SQL*Plus truncates the display of a SELECTed row if it is too long for the current line width.
SET XMLOPT[IMIZATIONCHECK] [ON|OFF]	Specifies that only fully optimized XML queries and DML operations are executed. Only to assist in developing and debugging, not for production.
SET XQUERY BASEURI {text}	Defines the base URI to use. This is useful to change the prefix of the file to access when writing generic XQuery expressions.
SET XQUERY ORDERING {UNORDERED | ORDERED | DEFAULT}	Controls the ordering of results from an XQuery.

System Variable	Description
SET XQUERY NODE {BYVALUE | BYREFERENCE | DEFAULT}	Sets the preservation mode for notes created or returned.
SET XQUERY CONTEXT {text}	Specifies an XQuery context item which can be either a node or a value.

13.45.1 SET APPINFO

Syntax

```
SET APPI[NFO] {ON | OFF | text}
```

Sets automatic registering of scripts through the DBMS_APPLICATION_INFO package.

This enables the performance and resource usage of each script 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 scripts invoked by the @, @@ or START commands. OFF disables registering of scripts. Instead, the current value of text is registered. *text* specifies the text to register when no script is being run or when APPINFO is OFF, which is the default. 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 script; *x* is '<' when the script name is truncated, otherwise, it is blank; and *filename* is the script name, possibly truncated to the length allowed by the DBMS_APPLICATION_INFO package interface.

Example

To display the value of APPINFO, as it is SET OFF by default, enter

```
SET APPINFO ON
SHOW APPINFO
```

```
APPINFO is ON and set to "SQL*Plus"
```

To change the default text, enter

```
SET APPINFO 'This is SQL*Plus'
```

To make sure that registration has taken place, enter

```
VARIABLE MOD VARCHAR2(50)
VARIABLE ACT VARCHAR2(40)
EXECUTE DBMS_APPLICATION_INFO.READ_MODULE(:MOD, :ACT);


PL/SQL procedure successfully completed.


PRINT MOD


MOD
--------------------------------------------------
This is SQL*Plus
```

To change APPINFO back to its default setting, enter

```
SET APPINFO OFF
```

13.45.2 SET ARRAYSIZE

Syntax

```
SET ARRAY[SIZE] {15 | n}
```

Sets the number of rows 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.

13.45.3 SET AUTOCOMMIT

Syntax

```
SET AUTO[COMMIT] {ON | OFF | IMM[EDIATE] | n}
```

Controls when Oracle Database commits pending changes to the database after SQL or PL/SQL commands.

ON commits pending changes to the database after Oracle Database executes each successful INSERT, UPDATE, or DELETE, 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 ON. n commits pending changes to the database after Oracle Database 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, or a SET AUTOCOMMIT command.

SET AUTOCOMMIT does not alter the commit behavior when SQL*Plus exits. Any uncommitted data is committed by default.

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

13.45.4 SET AUTOPRINT

Syntax

```
SET AUTOP[RINT] {ON | OFF}
```

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).

See PRINT for more information about displaying bind variables.

13.45.5 SET AUTORECOVERY

Syntax

```
SET AUTORECOVERY [ON | OFF]
```

ON sets the RECOVER command to automatically apply the default filenames of archived redo log files needed during recovery.

No interaction is needed, provided the necessary files are in the expected locations with the expected names. The filenames used are derived from the values of the initialization parameters LOG_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT.

OFF, the default option, requires that you enter the filenames manually or accept the suggested default filename given. See RECOVER for more information about database recovery.

Example

To set the recovery mode to AUTOMATIC, enter

```
SET AUTORECOVERY ON
RECOVER DATABASE
```

13.45.6 SET AUTOTRACE

Syntax

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
```

Displays a report on the execution of successful SQL DML statements (such as SELECT, INSERT, UPDATE, DELETE or MERGE).

The report can include execution statistics and the query execution path.

SQL*Plus report output may differ for DML if dynamic sampling is in effect.

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. See EXPLAIN PLAN for more information about EXPLAIN PLAN.

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.

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. The additional information and tabular output of AUTOTRACE PLAN is supported when connecting to Oracle Database 10*g* (Release 10.1) or later. When you connect to an earlier database, the older form or AUTOTRACE reporting is used.

AUTOTRACE is not available when FIPS flagging is enabled.

See About Tracing Statements for more information on AUTOTRACE.

13.45.7 SET BLOCKTERMINATOR

Syntax

```
SET BLO[CKTERMINATOR] {. | c | ON | OFF}
```

Sets the character used to end PL/SQL blocks to *c*.

It cannot be an alphanumeric character or a whitespace. To execute the block, you must issue a RUN or / (slash) command.

OFF means that SQL*Plus recognizes no PL/SQL block terminator. ON changes the value of *c* back to the default period (.), not the most recently used character.

13.45.8 SET CMDSEP

Syntax

```
SET CMDS[EP] {; | c | ON | OFF}
```

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 (;).

Example

To specify a title with TTITLE and format a column with COLUMN, both on the same line, enter

```
SET CMDSEP +
TTITLE LEFT 'SALARIES' + COLUMN SALARY FORMAT $99,999
SELECT LAST_NAME, SALARY FROM EMP_DETAILS_VIEW
WHERE JOB_ID = 'SH_CLERK';
```

```
SALARIES
LAST_NAME                  SALARY
------------------------- --------
Taylor                     $3,200
Fleaur                     $3,100
Sullivan                   $2,500
Geoni                      $2,800
Sarchand                   $4,200
Bull                       $4,100
Dellinger                  $3,400
Cabrio                     $3,000
Chung                      $3,800
Dilly                      $3,600
Gates                      $2,900
Perkins                    $2,500
Bell                       $4,000
Everett                    $3,900
McCain                     $3,200
Jones                      $2,800

SALARIES
LAST_NAME                  SALARY
------------------------- --------
Walsh                      $3,100
Feeney                     $3,000
OConnell                   $2,600
Grant                      $2,600

20 rows selected.
```

13.45.9 SET COLINVISIBLE

Syntax

```
SET COLINVI[SIBLE] [ON | OFF]
```

ON sets the DESCRIBE command to enable the display of information about an invisible column.

SET COLINVISIBLE has no effect on query statments that contain invisible columns. To retrieve data in an invisible column, explicitly specify the column in your query.

Example

To view information about an invisible column with the DESCRIBE command.

Create a table with an invisible column.

```
create table test_invisible_cols (emp_id number, emp_info char(20),
                                   emp_acc_no number invisible);
```

```
Table created.
```

Use the DESCRIBE command to list the table columns.

```
describe test_invisible_cols
```

```
Name                                     Null?    Type
---------------------------------------- -------- ------------------------
EMP_ID                                             NUMBER
EMP_INFO                                           CHAR(20)
```

Note that with the default SET COLINVISIBLE OFF, the invisible column does not appear in the result. Change the default setting of SET COLINVISIBLE to ON.

```
SET COLINVISIBLE ON
```

```
colinvisible ON
```

Now use the DESCRIBE command again to list the table columns. The invisible column now appears in the output.

```
describe test_invisible_cols
```

```
Name                                     Null?    Type
---------------------------------------- -------- ------------------------
EMP_ID                                             NUMBER
EMP_INFO                                           CHAR(20)
EMP_ACC_NO(INVISIBLE)                              NUMBER
```

13.45.10 SET COLSEP

Syntax

```
SET COLSEP {_ | text}
```

Sets the column separator character printed between columns in output.

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 {WR[APPED] | EA[CH] | OFF} for more information.

Example

To set the column separator to "|" enter

```
SET MARKUP HTML PREFORMAT ON
SET COLSEP '|'
SELECT LAST_NAME, JOB_ID, DEPARTMENT_ID
FROM EMP_DETAILS_VIEW
WHERE DEPARTMENT_ID = 20;


LAST_NAME                |JOB_ID    |DEPARTMENT_ID
-------------------------|----------|-------------
Hartstein                |MK_MAN    |           20
Fay                      |MK_REP    |           20
```

13.45.11 SET CONCAT

Syntax

```
SET CON[CAT] {. | c | ON | OFF}
```

Sets the character used to terminate a substitution variable reference when SQL*Plus would otherwise interpret the next character as a part of the variable name.

SQL*Plus resets the value of CONCAT to a period when you switch CONCAT on.

13.45.12 SET COPYCOMMIT

Syntax

```
SET COPYC[OMMIT] {0 | n}
```

Controls the number of rows 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.

13.45.13 SET COPYTYPECHECK

Syntax

```
SET COPYTYPECHECK {ON | OFF}
```

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.

13.45.14 SET DEFINE

Syntax

```
SET DEF[INE] {& | c | ON | OFF}
```

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.

See SET SCAN {ON|OFF} (obsolete) for more information on the SCAN variable.

13.45.15 SET DESCRIBE

Syntax

```
SET DESCRIBE [DEPTH {1 | n | ALL}] [LINENUM {ON | OFF}] [INDENT {ON | OFF}]
[[ANNO]TATION {ON|OFF}]
```

Sets the depth of the level to which you can recursively describe an object.

The valid range for the `DEPTH` clause is from 1 to 50. If you use `SET DESCRIBE DEPTH ALL`, then the depth will be set to 50, which is the maximum level allowed. You can display the line number and indentation of the attribute or column name when an object contains multiple object types. Use the `SET LINESIZE` command to control the width of the data displayed.

The `SET DESCRIBE` command is also used to enable and disable the display of annotations associated with columns. By default, annotation is disabled. When it is disabled, the `DESCRIBE` command does not display the annotation information. When annotation is enabled, the annotation information (if available for the column) is displayed. The default `linesize` is set to 120 characters. Users can reset the `linesize` as per their requirements.

If annotation data is longer than the length of the defined `ANNOTATIONS` column, then the data is aligned and wrapped to fit in the `ANNOTATIONS` column. The length of the `ANNOTATIONS` column is determined as a percentage of the `SET LINESIZE` value proportional to the column name and data type.

See DESCRIBE for more information about describing objects.

Example 1

The following example displays the use of the `SET DESCRIBE` command to describe an object up to the specified depth level. An object type `ADDRESS` is created:

```
CREATE TYPE ADDRESS AS OBJECT
  ( STREET  VARCHAR2(20),
    CITY    VARCHAR2(20)
```

```
  );
/
```

```
Type created
```

An EMPLOYEE table has a nested object, EMPADDR, of type ADDRESS as shown in the following example:

```
CREATE TABLE EMPLOYEE
  (LAST_NAME VARCHAR2(30),
   EMPADDR ADDRESS,
   JOB_ID VARCHAR2(20),
   SALARY NUMBER(7,2)
  );
/
```

```
Table created
```

You can use the SET DESCRIBE command to describe the table EMPLOYEE to a depth of two levels, to indent the output, and to display line numbers, as shown in the following example:

```
SET DESCRIBE DEPTH 2 LINENUM ON INDENT ON
DESCRIBE employee
```

The following output is displayed:

```
       Name                           Null?    Type
       ------------------------------ --------
--------------------------
1      LAST_NAME                               VARCHAR2(30)
2      EMPADDR                                 ADDRESS
3   2    STREET                                VARCHAR2(20)
4   2    CITY                                  VARCHAR2(20)
5      JOB_ID                                  VARCHAR2(20)
6      SALARY                                  NUMBER(7,2)
```

Example 2

The following example displays annotation information for a column. The table ANNOTATION_TAB has annotation information associated with the column c1:

```
CREATE TABLE annotation_tab
  (c1 NUMBER ANNOTATIONS(EmpGroup2 'Emp_Info', Hidden),
   c2 NUMBER PRIMARY KEY);
```

You can use the SET_DESCRIBE command to enable the display of annotation information for the column, as shown in the following example:

```
SET DESCRIBE ANNOTATION ON
DESCRIBE annotation_tab
```

The following output is displayed:

```
 Name                           Null?    Type
```

```
Annotations (Key: Value)
-------------------------- -------- -------------------------
--------------------------
C1                                        NUMBER                     EMPGROUP2:
Emp_Info
                                                                     HIDDEN:
C2                                        NUMBER
```

The table `ANNOTATION_TAB1` has annotation information associated with the column `c2`:

```
CREATE TABLE annotation_tab1
  (c1 NUMBER,
   c2 NUMBER ANNOTATIONS (EmpGroup1 'Emp_Info',
            EmpGroup2 'Emp_Info', Hidden));
```

You can use the `SET_DESCRIBE` command to enable the display of annotation information for the column, as shown in the following example:

```
SET DESCRIBE ANNOTATION ON
DESCRIBE annotation_tab1
```

The following output is displayed:

```
Name                        Null?    Type
Annotations (Key: Value)
-------------------------- -------- --------------------------
------------------------
C1                                    NUMBER
C2                                    NUMBER
EMPGROUP1: Emp_Info

EMPGROUP2: Emp_Info
                                                            HIDDEN:
```

Example 3

The following example displays the output for annotations associated with the table. The table `ANNOTATION_TAB2` has annotations associated with it:

```
CREATE TABLE annotation_tab2 (c1 NUMBER ANNOTATIONS(EmpGroup2 'Emp_Info', Hidden),
                  c2 NUMBER PRIMARY KEY)
                annotations (table_anno 'Table annotation 1',
                             table_anno2 'Table annotation 2'));
```

You can use the `SET_DESCRIBE` command to display the annotations associated with the table, as shown in the following example:

```
SET DESCRIBE ANNOTATION ON
DESCRIBE annotation_tab2
```

The following output is displayed:

```
Table Annotations (Key: Value)
-----------------------------------------------------------------------------
---------------
```

```
TABLE_ANNO: Table annotation 1
TABLE_ANNO2: Table annotation 2


Name                              Null?     Type
Annotations (Key: Value)
 -------------------------- -------- -------------------------
-------------------------------
 C1                                          NUMBER
EMPGROUP2: Emp_Info

HIDDEN:
 C2                                          NUMBER
```

13.45.16 SET ECHO

Syntax

```
SET ECHO {ON | OFF}
```

Controls whether or not to echo commands in a script that is executed with @, @@ or START. ON displays the commands on screen. OFF suppresses the display. ECHO does not affect the display of commands you enter interactively or redirect to SQL*Plus from the operating system.

13.45.17 SET EDITFILE

Syntax

```
SET EDITF[ILE] file_name[.ext]
```

Sets the default filename for the EDIT command. See EDIT for more information about the EDIT command. The default filename for the EDIT command is afiedt.buf which is the SQL buffer.

You can include a path and/or file extension. See SET SUF[FIX] {SQL | text} for information on changing the default extension. The default filename and maximum filename length are operating system specific.

13.45.18 SET EMBEDDED

Syntax

```
SET EMB[EDDED] {ON | OFF}
```

Controls where on a page each report begins.

OFF forces each report to start at the top of a new page. ON enables 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.

13.45.19 SET ERRORDETAILS

Syntax

```
SET ERRORDETAILS { OFF | ON | VERBOSE }
```

Enables SQL*Plus to display the Oracle Database Error Help URL, along with the cause and action details, when any SQL, PL/SQL, or SQL*Plus statement fails during execution.

Setting ERRORDETAILS to ON displays the Oracle Database Error Help URL. This is the default setting. Setting ERRORDETAILS to VERBOSE displays the Oracle Database Error Help URL and additional details, such as the error message cause, action, and parameters. Setting it to OFF neither displays the URL nor other error help details.

> **Note:**
>
> The default value of ERRORDETAILS is ON. When the environment variable ORA_SUPPRESS_ERROR_URL is set, the value ERRORDETAILS is set to ON or OFF based on the value set in the ORA_SUPPRESS_ERROR_URL variable.

Examples

In the following example, the variable ERRORDETAILS is set to ON:

```
SET ERRORDETAILS ON
SELECT * FROM EMP;
SP2-0640: Not connected
Help: https://docs.oracle.com/error-help/db/sp2-0640/
```

In the following example, the variable ERRORDETAILS is set to OFF:

```
SET ERRORDETAILS OFF
DECLARE
  B VARCHAR2(100);
BEGIN
  SELECT EMPNO INTO B FROM EMP;
END;
/
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returned more than the requested number of rows (1)
```

In the following example, the variable ERRORDETAILS is set to VERBOSE:

```
SET ERRORDETAILS VERBOSE
DECLARE
  B VARCHAR2(100);
BEGIN
  SELECT EMPNO INTO B FROM EMP;
```

```
END;
/
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returned more than the requested number of rows
(1)
Help: https://docs.oracle.com/error-help/db/ora-01422/
Cause: The cause is one of the following:
    1. A SELECT statement was executed in the exact fetch
       mode and returned more rows than requested.
    2. In PL/SQL, a SELECT INTO statement returned more than one row.
    3. In PL/SQL, a DML RETURNING INTO statement
       returned more than one row.
Action: Choose the action corresponding to the cause as numbered:
    1. Increase the number of rows requested to accommodate the
       number of rows returned;
       or omit the exact fetch mode on the fetch call.
    2. In PL/SQL, use a FOR loop to process the rows.
    3. In PL/SQL, use BULK COLLECT to return values into a table.
Params: 1) requested_rows: The number of requested rows.
```

13.45.20 SET ERRORLOGGING

Syntax

```
SET ERRORL[OGGING] {ON | OFF} [TABLE [schema.]tablename] [TRUNCATE]
[IDENTIFIER identifier]
```

Turns SQL*Plus error logging ON or OFF. Error logging records SQL, PL/SQL, and SQL*Plus errors and associated parameters in an error log table. You can then query the log table to review errors resulting from a query. When error logging is ON, errors are recorded, whether the query is run interactively or from a script. This is particularly useful for capturing errors generated from long running queries and avoids capturing all output using the SPOOL command or having to remain present during the run.

By default, errors are written to a the table SPERRORLOG in your schema. If this table does not exist, it is created automatically. You can also use the TABLE *schema.tablename* option to specify other tables to use. When using a table other than SPERRORLOG, it must already exist, and you must have access to it. See Creating a User Defined Error Log Table.

If an internal error occurs, to avoid recursion errors caused by the errorlog calling itself, errorlogging is automatically set OFF.

Error logging is set OFF by default.

```
ON
```

Writes ORA, PLS and SP2 errors to the default table, SPERRORLOG.

```
OFF
```

Turns off error logging.

```
TABLE [schema.]tablename
```

Specifies a user defined table to use instead of the default, SPERRORLOG. If you omit *schema.* the table is created in the current schema. The table you specify must exist, and you must have access permissions.

If the table specified does not exist, or you do not have access, an error message is displayed and the default table, SPERRORLOG, is used.

```
TRUNCATE
```

Clears all existing rows in the error log table and begins recording errors from the current session.

```
IDENTIFIER identifier
```

A user defined string to identify errors. You can use it to identify errors from a particular session or from a particular version of a query.

Creating a User Defined Error Log Table

You can create one or more error log tables to use other than the default. Before specifying a user defined error log table with the TABLE *schema.tablename* option, you must create it and ensure that you have permissions to access it. The error log table has the following column definitions:

Table 13-4   SQL*Plus Error Log Column Definitions

Column	Type	Description
username	VARCHAR(256)	Oracle account name.
timestamp	TIMESTAMP	Time when the error occurred.
script	VARCHAR(1024)	Name of the originating script if applicable.
identifier	VARCHAR(256)	User defined identifier string.
message	CLOB	ORA, PLA or SP2 error message. No feed back messages are included. For example, "PL/SQL Block Created" is not recorded.
statement	CLOB	The statement causing the error.

Using User Defined Error Log Tables

To use a user defined log table, you must have permission to access the table, and you must issue the SET ERRORLOGGING command with the TABLE *schema.tablename* option to identify the error log table and the schema if applicable.

Querying Error Log Tables

To view the records recorded in an error log table, you query the columns you want to view as for any table. The columns available are shown in Table 13-4.

Example

To use the default error log table to record query errors from a script, *myfile.sql*, which contains the following:

```
VARIABLE U REFCURSOR
BEGIN
  OPEN :U FOR SELECT * FROM DEPT;
END;
/
```

```
SHOW ERRORS PROCEDURE 'SSS'

SET GARBAGE

SELECT *
FROM
GARBAGE
;
```

Enter the following:

```
SET ERRORLOGGING ON
@myfile
```

which produces the following output:

```
open :u for select * from dept;
                              *
ERROR at line 2:
ORA-6550: line 2, column 29:
PLS-00201: ORA-00942: table or view does not exist
ORA-6550: line 2, column 16:
PL/SQL: SQL Statement ignored

ERROR:
ORA-00907: missing right parenthesis

SP2-0158: unknown SET option "garbage"

garbage
*
ERROR at line 3:
ORA-00942: table or view does not exist
```

To view the errror log written to the default error log table, SPERRORLOG, enter:

```
SELECT TIMESTAMP, USERNAME, SCRIPT, IDENTIFIER, STATEMENT, MESSAGE
FROM SPERRORLOG;
```

which produces the following output:

TIMESTAMP	USERNAME	SCRIPT	IDENTIFIER	STATEMENT	MESSAGE
Mon May 08 21:30:03 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	ORA-06550: line 2, column 27:
Mon May 08 21:30:05 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	PL/SQL: ORA-00942: table or view does not exist
Mon May 08 21:30:05 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	ORA-06550: line 2, column 13:
Mon May 08 21:30:05 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	PL/SQL: SQL Statement ignored

TIMESTAMP	USERNAME	SCRIPT	IDENTIFIER	STATEMENT	MESSAGE
Mon May 08 21:30:06 2006	SYSTEM	d:\myfile.sql		show errors procedure "sss"	ORA-00907: missing right parenthesis
Mon May 08 21:30:09 2006	SYSTEM	d:\myfile.sql		set garbage	SP2-0158: unknown SET option "garbage"
Mon May 08 21:30:10 2006	SYSTEM	d:\myfile.sql		garbage	ORA-00942: table or view does not exist

Example 2

To use a user defined error log table to record query errors from a script, *myfile.sql*, which contains the following:

```
VARIABLE U REFCURSOR
BEGIN
  OPEN :U FOR SELECT * FROM DEPT;
END;
/

SHOW ERRORS PROCEDURE 'SSS'

SET GARBAGE

SELECT *
FROM
GARBAGE
;
```

Enter the following:

```
SET ERRORLOGGING ON
@MYFILE
```

which produces the following output:

```
open :u for select * from dept;
                              *
ERROR at line 2:
ORA-6550: line 2, column 29:
PLS-00201: ORA-00942: table or view does not exist
ORA-6550: line 2, column 16:
PL/SQL: SQL Statement ignored

ERROR:
ORA-00907: missing right parenthesis

SP2-0158: unknown SET option "garbage"

garbage
*
ERROR at line 3:
ORA-00942: table or view does not exist
```

To view the errror log written to the default error log table, SPERRORLOG, enter:

```
SELECT TIMESTAMP, USERNAME, SCRIPT, IDENTIFIER, STATEMENT, MESSAGE
FROM SPERRORLOG;
```

which produces the following output:

TIMESTAMP	USERNAME	SCRIPT	IDENTIFIER	STATEMENT	MESSAGE
Mon May 08 21:30:03 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	ORA-06550: line 2, column 27:
Mon May 08 21:30:05 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	PL/SQL: ORA-00942: table or view does not exist
Mon May 08 21:30:05 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	ORA-06550: line 2, column 13:
Mon May 08 21:30:05 2006	SYSTEM	d:\myfile.sql		open :u for select * from dept;	PL/SQL: SQL Statement ignored
Mon May 08 21:30:06 2006	SYSTEM	d:\myfile.sql		show errors procedure "sss"	ORA-00907: missing right parenthesis
Mon May 08 21:30:09 2006	SYSTEM	d:\myfile.sql		set garbage	SP2-0158: unknown SET option "garbage"
Mon May 08 21:30:10 2006	SYSTEM	d:\myfile.sql		garbage	ORA-00942: table or view does not exist

Example 3

To use an error log table other than the default:

- Create the error log table you want to use
- Specify the table you want to use with the TABLE option of the SET ERRORLOGGING ON command.

The error log table must have the column definitions defined in Table 13-4.

John wants to use an error log table named *john_sperrorlog*. John would run the following SQL statements to create the new error log table:

```
DROP TABLE john_sperrorlog;
CREATE TABLE john_sperrorlog(username   VARCHAR(256),
timestamp  TIMESTAMP,
script     VARCHAR(1024),
identifier VARCHAR(256),
message    CLOB,
statement  CLOB);
```

Removed Commit from previous example from user comment giridhar123 14Feb08

John then issues the following SET command to enable error logging using the newly created error log table

```
SET ERRORLOGGING ON TABLE john_sperrorlog
```

All error logging for John is now recorded to *john_sperrorlog*, and not to the default error log table, SPERRORLOG.

Access privileges for the error log table are handled in the same way as for any user table.

13.45.21 SET ESCAPE

Syntax

```
SET ESC[APE] {\ | c | ON | OFF}
```

Defines the character used as the escape character.

OFF undefines the escape character. ON enables the escape character. ON changes the value of *c* back to the default "\".

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.

Example

If you define the escape character as an exclamation point (!), then

```
SET ESCAPE !
ACCEPT v1 PROMPT 'Enter !&1:'
```

displays this prompt:

```
Enter &1:
```

To set the escape character back to the default value of \ (backslash), enter

```
SET ESCAPE ON
```

13.45.22 SET ESCCHAR

Syntax

```
SET ESCCHAR {@ | ? | % | OFF}
```

Specifies a character to be escaped and not interpreted when used in a file name for the SPOOL, START, @, RUN and EDIT commands. These special characters are translated to the following:

• @ in a filename will be translated to Oracle SID

• ? is translated to Oracle Home in Unix

• % is translated to Oracle Home in Windows

While it is not recommended that these characters are used in filenames, if you have legacy files that do use them, it might be useful to include a SET ESCCHAR command in your GLogin file to implement it across your site.

If not escaped, the characters @, ? and % have significance when interpreted and cause errors for the SPOOL, START, @, RUN and EDIT commands.

SET ESCCHAR is set OFF by default.

> **✎ Note:**
>
> Starting from Oracle Database release 19c, version 19.3, file names with the $ character will no longer run on Windows.

13.45.23 SET EXITCOMMIT

Syntax

```
SET EXITC[OMMIT] {ON | OFF}
```

Specifies whether the default EXIT behavior is COMMIT or ROLLBACK.

The default setting is ON, which means that work is committed on exit, whether you expected it to be committed or not. Set EXITCOMMIT OFF to rollback work on exit.

Table 13-5 shows the exit action associated with combinations of SET commands (AUTOCOMMIT & EXITCOMMIT) and the EXIT command.

Table 13-5   Exit Behavior: AUTOCOMMIT, EXITCOMMIT, EXIT

AUTOCOMMIT	EXITCOMMIT	EXIT	Exit Behavior
ON	ON	-	COMMIT
ON	OFF	-	COMMIT
OFF	ON	-	COMMIT
OFF	OFF	-	ROLLBACK
ON	ON	COMMIT	COMMIT
ON	ON	ROLLBACK	COMMIT
ON	OFF	COMMIT	COMMIT
ON	OFF	ROLLBACK	COMMIT
OFF	ON	COMMIT	COMMIT
OFF	ON	ROLLBACK	ROLLBACK
OFF	OFF	COMMIT	COMMIT
OFF	OFF	ROLLBACK	ROLLBACK

13.45.24 SET FEEDBACK

Syntax

```
SET FEED[BACK] {6 | n | ON | OFF | ONLY} [SQL_ID]
```

Displays the number of records returned by a script when a script 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. The feedback message is not displayed while the data is displayed.

SET FEEDBACK OFF also turns off the statement confirmation messages such as 'Table created' and 'PL/SQL procedure successfully completed' that are displayed after successful SQL or PL/SQL statements.

ONLY returns the number of rows selected by a query without displaying data.

SQL_ID returns the sql_id for the SQL or PL/SQL statements that are executed. The sql_id will be assigned to the predefined variable _SQL_ID. You can use this predefined variable to debug the SQL statement that was executed. The variable can be used like any other predefined variable, such as _USER and _DATE.

```
SQL> SET FEEDBACK ON SQL_ID
SQL> SELECT * FROM DUAL;

D
-
X

1 row selected.

SQL_ID: a5ks9fhw2v9s1
SQL> COLUMN sql_text FORMAT a50
SQL> SELECT sql_text FROM v$sql WHERE sql_id = '&_sql_id';

SQL_TEXT
--------------------------------------------------
SELECT * FROM DUAL

1 row selected.

SQL_ID: cf9bgxbfytv5b
```

When the SQL_ID option is specified and feedback is ON, you see the sql id displayed along with the feedback message. When feedback is OFF, only the sql id is displayed.

Example

To enable SET FEEDBACK ONLY, enter

```
SQL> SET FEEDBACK ONLY
SQL> SHOW FEEDBACK
feedback ONLY
SQL> SELECT * FROM EMP;

14 rows selected.
```

To enable SET_FEEDBACK and SQL_ID enter

```
SQL> SET FEEDBACK ON SQL_ID
SQL> SELECT * FROM DEPT;

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

```
4 rows selected.

SQL_ID: 3154rqzb8xudy
SQL> CONNECT SYSTEM
ENTER PASSWORD:
Connected.
SQL> COLUMN sql_text FORMAT a50
SQL> SELECT sql_text FROM v$sql WHERE sql_id = '&_sql_id';

SQL_TEXT
-----------------------------------------------------------
SELECT * FROM DEPT

1 row selected.
SQL_ID: 81a5n8q6g2vvr
```

13.45.25 SET FLAGGER

Syntax

```
SET FLAGGER {OFF | 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 Database 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.

13.45.26 SET FLUSH

Syntax

```
SET FLU[SH] {ON | OFF}
```

Controls when output is sent to the user's display device. OFF enables the operating system to buffer output. ON disables buffering. FLUSH only affects display output, it does not affect spooled output.

Use OFF only when you run a script non-interactively (that is, when you do not need to see output and/or prompts until the script finishes running). The use of FLUSH OFF may improve performance by reducing the amount of program I/O.

13.45.27 SET HEADING

Syntax

```
SET HEA[DING] {ON | OFF}
```

Controls printing of column headings in reports.

ON prints column headings in reports; OFF suppresses column headings.

The SET HEADING OFF command does not affect the column width displayed, it only suppresses the printing of the column header itself.

Example

To suppress the display of column headings in a report, enter

```
SET HEADING OFF
```

If you then run a SQL SELECT command

```
SELECT LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE JOB_ID = 'AC_MGR';
```

the following output results:

```
Higgins 12000
```

To turn the display of column headings back on, enter

```
SET HEADING ON
```

13.45.28 SET HEADSEP

Syntax

```
SET HEADS[EP] {_ | c | ON | OFF}
```

Defines the character used as a line break in column headings.

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 "|".

13.45.29 SET HISTORY

Syntax

```
SET HIST[ORY] {ON | OFF | n}
```

Enables or disables the history of commands. When enabled, SQL*Plus, SQL and PL/SQL statements are stored in the command history list. You can recall SQL*Plus, SQL and PL/SQL statements by using the HISTORY command.

Examples

To enable command history and store 200 entries in the list, enter

```
SET HIST[ORY] 200
```

If you do not specify a value for *n*, the default is 100.

> **Note:**
>
> Multiline entries such as PL/SQL blocks are considered as one single entry in the command history list. You can store up to 100000 commands in command history. If you try to enable command history while it is already enabled, the old command history is cleared.

13.45.30 SET INSTANCE

Syntax

```
SET INSTANCE [instance_path | LOCAL]
```

Changes the default instance for your session to the specified instance path.

Using the SET INSTANCE command does not connect to a database. The default instance is used for commands when no instance is specified. Any commands preceding the first use of SET INSTANCE communicate with the default instance.

To reset the instance to the default value for your operating system, you can either enter SET INSTANCE with no *instance_path* or SET INSTANCE LOCAL.

Note, you can only change the instance when you are not currently connected to any instance. That is, you must first make sure that you have disconnected from the current instance, then set or change the instance, and reconnect to an instance in order for the new setting to be enabled.

This command may only be issued when Oracle Net is running. You can use any valid Oracle Net connect identifier as the specified instance path. See your operating system-specific Oracle Database documentation for a complete description of how your operating system specifies Oracle Net connect identifiers. The maximum length of the instance path is 64 characters.

Example

To set the default instance to "PROD1" enter

```
DISCONNECT
SET INSTANCE PROD1
```

To set the instance back to the default of local, enter

```
SET INSTANCE local
```

You must disconnect from any connected instances to change the instance.

13.45.31 SET JSONPRINT

Syntax

```
SET JSONPRINT [NORMAL] [PRETTY] [ASCII]
```

Formats the output of JSON type columns.

Starting with Release 21c, JSON is a new SQL and PL/SQL data type for JSON data. For more information about JSON, see Getting Started with JSON.

PRETTY displays a formatted JSON output with proper alignment and spacing.

ASCII displays non-ASCII characters as Unicode escape sequences.

NORMAL is the default value and clears the PRETTY and ASCII parameters.

Example

To enable pretty printing for JSON data, enter:

```
SQL> set jsonprint pretty
SQL> show jsonprint
jsonprint PRETTY

SQL> select json('[{a:"\u20ac"},{b:"value"}]') from dual;
JSON('[{A:"\U20AC"},{B:"VALUE"}]')
-------------------------------------------------------------
---------------
[
 {
 "a" : "€"
 },
 {
 "b" : "value"
 }
]
```

The result has a non-ASCII character. If ASCII is enabled, the output is as follows:

```
SQL> set jsonprint pretty ascii
SQL> show jsonprint
jsonprint PRETTY ASCII

SQL> select json('[{a:"\u20ac"},{b:"value"}]') from dual;
JSON('[{A:"\U20AC"},{B:"VALUE"}]')
------------------------------------------------------------
---------------
[
 {
 "a" : "\u20AC"
 },
 {
 "b" : "value"
 }
]
```

13.45.32 SET LINESIZE

Syntax

```
SET LIN[ESIZE] {80 | n | WINDOW}
```

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. Changing the linesize setting can affect text wrapping in output from the DESCRIBE command. DESCRIBE output columns are typically allocated a proportion of the linesize. Decreasing or increasing the linesize may give unexpected text wrapping in your display. You can define LINESIZE as a value from 1 to a maximum that is system dependent.

WINDOW adjusts the linesize and pagesize for the formatted output according to the width and height of the screen. If the output is longer than the screen width, then the output is wrapped accordingly.

If the screen size is manually set by using the SET LINESIZE *n* command, the subsequent output will be displayed to fit the new linesize.

> **Note:**
>
> The SET LINESIZE WINDOW command is ignored in a script and the output will not be displayed according to the screen size.

Example

To set the linesize for the output to 20, enter

```
SQL> SET LINESIZE 20
SQL> SHOW LINESIZE
linesize 20
```

To dynamically change the output display after manually resizing the screen, enter

```
SQL> SET LINESIZE Window
SQL> SHOW LINESIZE
linesize 160 WINDOW
```

13.45.33 SET LOBOFFSET

Syntax

```
SET LOBOF[FSET] {1 | n}
```

Sets the starting position from which BLOB, BFILE, CLOB and NCLOB data is retrieved and displayed.

Example

To set the starting position from which a CLOB column's data is retrieved to the 22nd position, enter

```
SET LOBOFFSET 22
```

The CLOB data will wrap on your screen; SQL*Plus will not truncate until the 23rd character.

13.45.34 SET LOBPREFETCH

Syntax

```
SET LOBPREFETCH {0 | n}
```

Sets the amount of LOB data (in bytes) that SQL*Plus will prefetch from the database at one time.

Example

To set the amount of prefetched LOB data to 8000 bytes, enter

```
SET LOBPREFETCH 8000
```

If you do not specify a value for *n*, the default is 0. This means that LOB data prefetching is off.

> ✎ **Note:**
>
> You can specify a maximum value of 2147483648 bytes (2 Gigabytes). The settings in the `oraaccess.xml` file can override the SET LOBPREFETCH setting in SQL*Plus. For more information about `oraaccess.xml`, see the *Oracle Call Interface Programmer's Guide*.

To show the current setting for the amount of LOB data that SQL*Plus will prefetch from the database at one time, enter

```
SHOW LOBPREF[ETCH]
```

13.45.35 SET LOGSOURCE

Syntax

```
SET LOGSOURCE [pathname]
```

Specifies the location from which archive logs are retrieved during recovery.

The default value is set by the LOG_ARCHIVE_DEST initialization parameter in the Oracle Database initialization file, init.ora. Using the SET LOGSOURCE command without a *pathname* restores the default location.

Example

To set the default location of log files for recovery to the directory "/usr/oracle10/dbs/arch" enter

```
SET LOGSOURCE "/usr/oracle10/dbs/arch"
RECOVER DATABASE
```

13.45.36 SET LONG

Syntax

```
SET LONG {80 | n}
```

Sets maximum width (in bytes) for displaying BLOB, BFILE, CLOB, LONG, NCLOB and XMLType values; and for copying LONG values.

Starting with Oracle Database release 21c, `JSONPRINT` can be used with `SET LONG` to set the maximum width for displaying JSON values.

Querying LONG columns requires enough local memory to store the amount of data specified by SET LONG, irrespective of the value of the SET LONGCHUNKSIZE command. This requirement does not apply when querying LOBs.

It is recommended that you do not create tables with LONG columns. LONG columns are supported only for backward compatibility. Use LOB columns (BLOB, BFILE, CLOB, NCLOB) instead. LOB columns have fewer restrictions than LONG columns and are still being enhanced.

The maximum value of *n* is 2,000,000,000 bytes. It is important to check that the memory required by your SET LONG command is available on your machine, for example:

```
SET LONG 2000000000
```

assumes that available RAM (random access memory) on the machine exceeds 2 gigabytes.

Example

To set the maximum number of bytes to fetch for displaying and copying LONG values, to 500, enter

```
SET LONG 500
```

The LONG data will wrap on your screen; SQL*Plus will not truncate until the 501st byte. The default for LONG is 80 bytes.

13.45.37 SET LONGCHUNKSIZE

Syntax

```
SET LONGC[HUNKSIZE] {80 | n}
```

Sets the size (in bytes) of the increments SQL*Plus uses to retrieve a BLOB, BFILE, CLOB, LONG, NCLOB or XMLType value.

LONGCHUNKSIZE is not used for object relational queries such as CLOB, or NCLOB.

Example

To set the size of the increments in which SQL*Plus retrieves LONG values to 100 bytes, enter

```
SET LONGCHUNKSIZE 100
```

The LONG data will be retrieved in increments of 100 bytes until the entire value is retrieved or the value of SET LONG is reached, whichever is the smaller.

13.45.38 SET MARKUP

Syntax

```
SET MARK[UP] markup_option
```

where *markup_option* consists of:

• *csv_option*

• *html_option*

where *csv_option* has the following syntax:

```
CSV {ON|OFF} [DELIMI[TER] character] [QUOTE {ON|OFF}]
```

where *html_option* has the following syntax:

```
HTML {ON|OFF} [HEAD text] [BODY text] [TABLE text] [ENTMAP {ON|OFF}] [SPOOL {ON|OFF}]
[PRE[FORMAT] {ON|OFF}]
```

csv_option

Outputs reports in CSV format.

To be effective, SET MARKUP commands that change values in dynamic report output must be issued before the statement that produces the query output. The first statement that

produces the query output triggers the output of CSV data that reflects the DELIMITER and QUOTE settings.

CSV is a mandatory SET MARKUP argument which specifies the type of output to be generated is CSV. The optional CSV arguments, ON and OFF, specify whether or not to generate CSV output. The default is OFF.

You can turn CSV output ON and OFF as required during a session.

```
DELIMI[TER] character
```

The DELIMI[TER] *character* option enables you to specify a column separator character.

```
QUOTE {ON|OFF}
```

The QUOTE {ON|OFF} option enables you to turn text quoting on or off. The default is OFF.

QUOTE ON generates CSV output with all text quoted. Double quotes (" ") embedded within text are escaped.

You can turn quoted text ON and OFF as required during a session.

Supported Commands when SET MARKUP CSV is Enabled

If enabled, the following COLUMN commands will remain effective when SET MARKUP CSV is enabled:

- COLUMN FORMAT
- COLUMN HEADING
- COLUMN NULL

Unsupported Commands when SET MARKUP CSV is Enabled

When SET MARKUP CSV is enabled, the following SQL*Plus commands will have no effect on the output:

- BREAK
- BTITLE
- COMPUTE
- REPFOOTER
- REPHEADER

When SET MARKUP CSV is enabled, the following SET commands will have no effect on the output:

- SET COLSEP
- SET HEADSEP
- SET LINESIZE
- SET NEWPAGE
- SET PAGESIZE

- [SET PAUSE](#)
- [SET RECSEP](#)
- [SET SHIFTINOUT](#)
- [SET TAB](#)
- [SET TRIMOUT](#)
- [SET TRIMSPOOL](#)
- [SET UNDERLINE](#)
- [SET WRAP](#)

When SET MARKUP CSV is enabled, the following COLUMN commands will have no effect on the output:

- COLUMN ENTMAP
- COLUMN FOLD_AFTER
- COLUMN FOLD_BEFORE
- COLUMN JUSTIFY
- COLUMN NEWLINE
- COLUMN NEW_VALUE
- COLUMN NOPRINT
- COLUMN OLD_VALUE
- COLUMN WRAP

Use the SHOW MARKUP command to view the status of MARKUP options.

Example

The following example illustrates the output when SET MARKUP CSV is enabled:

```
SQL> SET MARKUP CSV ON
SQL> SELECT * FROM EMP;

"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,,10
7788,"SCOTT","ANALYST",7566,"19-APR-87",3000,,20
7839,"KING","PRESIDENT",,"17-NOV-81",5000,,10
7844,"TURNER","SALESMAN",7698,"08-SEP-81",1500,0,30
7876,"ADAMS","CLERK",7788,"23-MAY-87",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

14 rows selected.
```

The following example illustrates how to extract all records from the Employee table of the database, with text strings unquoted:

```
SQL> SET MARKUP CSV ON QUOTE OFF
SQL> SELECT * FROM EMP;

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,,10
7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20
7839,KING,PRESIDENT,,17-NOV-81,5000,,10
7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30
7876,ADAMS,CLERK,7788,23-MAY-87,1100,,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

14 rows selected.
```

The following example illustrates the output with the pipe (|) character specified as the delimiter:

```
SQL> SET MARKUP CSV ON DELIMITER |
SQL> SELECT * FROM EMP;

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||10
7788|SCOTT|ANALYST|7566|19-APR-87|3000||20
7839|KING|PRESIDENT||17-NOV-81|5000||10
7844|TURNER|SALESMAN|7698|08-SEP-81|1500|0|30
7876|ADAMS|CLERK|7788|23-MAY-87|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

14 rows selected.
```

html_option

Outputs HTML marked up text.

To be effective, SET MARKUP commands that change values in dynamic report output must occur before statements that produce query output. The first statement that produces query output triggers the output of information affected by SET MARKUP such as HEAD and TABLE settings. Subsequent SET MARKUP commands have no effect on the information already sent to the report.

SET MARKUP only specifies that SQL*Plus output will be HTML encoded. You must use SET MARKUP HTML ON SPOOL ON and the SQL*Plus SPOOL command to create and name a spool file, and to begin writing HMTL output to it. SET MARKUP has the same options and behavior as SQLPLUS -MARKUP.

See MARKUP Options for detailed information. For examples of usage, see Generating HTML Reports from SQL*Plus.

Use the SHOW MARKUP command to view the status of MARKUP options.

Example

The following is a script which uses the SET MARKUP HTML command to enable HTML marked up text to be spooled to a specified file:

> **Note:**
>
> The SET MARKUP example command is laid out for readability using line continuation characters "–" and spacing. Command options are concatenated in normal entry.

Use your favorite text editor to enter the commands necessary to set up the HTML options and the query you want for your report.

```
SET MARKUP HTML ON SPOOL ON HEAD "<TITLE>SQL*Plus Report</title> -
<STYLE TYPE='TEXT/CSS'><!--BODY {background: ffffc6} --></STYLE>"
SET ECHO OFF
SPOOL employee.htm
SELECT FIRST_NAME, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000;
SPOOL OFF
SET MARKUP HTML OFF
SET ECHO ON
```

As this script contains SQL*Plus commands, do not attempt to run it with / (slash) from the buffer because it will fail. Save the script in your text editor and use START to execute it:

```
START employee.sql
```

As well as writing the HTML spool file, employee.htm, the output is also displayed on screen because SET TERMOUT defaults to ON. You can view the spool file, employee.htm, in your web browser. It should appear something like the following:

FIRST_NAME	LAST_NAME	SALARY
Steven	King	24000
Neena	Kochhar	17000
Lex	De Haan	17000
Michael	Hartstein	13000
John	Russell	14000
Karen	Partners	13500

6 rows selected.

13.45.39 SET NEWPAGE

Syntax

```
SET NEWP[AGE] {1 | 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.

13.45.40 SET NULL

Syntax

```
SET NULL text
```

Sets the text displayed whenever a null value occurs 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. The default output for a null is blank ("").

13.45.41 SET NUMFORMAT

Syntax

```
SET NUMF[ORMAT] format
```

Sets the default format for displaying numbers. See the FORMAT clause of the COLUMN command for number format descriptions. Enter a number format for *format*. To use the default field width and formatting model specified by SET NUMWIDTH, enter

```
SET NUMFORMAT ""
```

13.45.42 SET NUMWIDTH

Syntax

```
SET NUM[WIDTH] {10 | n}
```

Sets the default width for displaying numbers. See the FORMAT clause of the COLUMN command for number format descriptions.

COLUMN FORMAT settings take precedence over SET NUMFORMAT settings, which take precedence over SET NUMWIDTH settings.

13.45.43 SET PAGESIZE

Syntax

```
SET PAGES[IZE] {14 | n}
```

Sets the number of lines on each page of output. You can set PAGESIZE to zero to suppress all headings, page breaks, titles, the initial blank line, and other formatting information.

13.45.44 SET PAUSE

Syntax

```
SET PAU[SE] {ON | OFF | text}
```

Enables you to control scrolling of your terminal when running reports. You need to first, SET PAUSE *text*, and then SET PAUSE ON if you want text to appear each time SQL*Plus pauses.

SET PAUSE ON pauses output at the beginning of each PAGESIZE number of lines of report output. Press Return to view more output. SET PAUSE *text* specifies the text to be displayed each time SQL*Plus pauses. Multiple words in *text* must be enclosed 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.

13.45.45 SET RECSEP

Syntax

```
SET RECSEP {WR[APPED] | EA[CH] | OFF}
```

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.

13.45.46 SET RECSEPCHAR

Syntax

```
SET RECSEPCHAR {_ | c}
```

Defines the character to display or print to separate records.

A record separator consists of a single line of the RECSEPCHAR (record separating character) repeated LINESIZE times. The default is a single space.

13.45.47 SET ROWLIMIT

Syntax

```
SET ROWLIMIT {n | OFF}
```

Sets a limit for the number of rows to display for a query. By default, ROWLIMIT is OFF.

n can be any number between 1 and 2,000,000,000. An error is displayed when the value that is entered is outside this range.

OFF displays all the rows in the output.

SET ROWLIMIT ignores the value set by the SET FEEDBACK command.

The SET ROWLIMIT command is useful for queries that return a large number of rows, where users want to limit the number of rows to display without changing the query.

The purpose of the SET ROWLIMIT command is not to improve the performance of a query but to limit the rows to display. The number of rows fetched from the database is determined by the SET ARRAYSIZE value. Therefore, the server may return more rows than the value set in ROWLIMIT. When that happens, the query will finish and will not fetch any more data from the server. For example, a SQL statement to query a table contains 100 rows, but ROWLIMIT is set to 10 and ARRAYSIZE is set to 15. When the query is executed, 15 rows will first be fetched from the database. Since the rows returned satisfy the ROWLIMIT setting, the query will finish without fetching any more rows. However, if the ROWLIMIT is set to 20, then a second round trip is required to fetch the next 15 rows to satisfy the ROWLIMIT setting.

Example

To set the number of rows to display as 2, enter:

```
SQL> SET ROWLIMIT 2
SQL> SHOW ROWLIMIT
rowlimit 2

SQL> SELECT * FROM DEPT;

DEPTNO  DNAME
------  -----------
10      ACCOUNTING
20      RESEARCH

2 rows selected. (rowlimit reached)
```

Example

By default, the FEEDBACK message is displayed only after 10 rows are displayed, but if ROWLIMIT is set to 2, the FEEDBACK message will be displayed for two rows even if the FEEDBACK is set to 10.

```
SQL> SET FEEDBACK 10
SQL> SET ROWLIMIT 2
```

```
SQL> SELECT * from EMP;

2 rows selected. (rowlimit reached)
```

13.45.48 SET ROWPREFETCH

Syntax

```
SET ROWPREFETCH {1 | n}
```

Sets the number of rows that SQL*Plus will prefetch from the database at one time.

The default value is 1.

Example

To set the number of prefetched rows to 200, enter

```
SET ROWPREFETCH 200
```

If you do not specify a value for *n*, the default is 1 row. This means that row prefetching is off.

> **Note:**
>
> The amount of data contained in the prefetched rows should not exceed the
> maximum value of 2147483648 bytes (2 Gigabytes). The `<prefetch>` setting in
> the `oraaccess.xml` file can override the SET ROWPREFETCH setting in SQL*Plus.
> For more information about `oraaccess.xml`, see the *Oracle Call Interface*
> *Programmer's Guide*.

To show the current setting for the number of rows that SQL*Plus will prefetch from the
database at one time, enter

```
SHOW ROWPREF[ETCH]
```

13.45.49 SET SECUREDCOL

Syntax

```
SET SECUREDCOL {OFF | ON} [UNAUTH[ORIZED] text] [UNK[NOWN] text]
```

Sets how secure column values are displayed in SQLPLUS output for users without
permission to view a column and for columns with unknown security. You can choose either
the default text or specify the text that is displayed. The default is OFF.

When column level security is enabled, and SET SECUREDCOL is set ON, output from
SQLPLUS for secured columns or columns of unknown security level is replaced with either
your customized text or the default indicators. This only applies to scalar data types. Complex
Object data output is not affected.

ON displays the default indicator "*****" in place of column values for users without
authorisation, and displays "?????" in place of column values where the security level is

unknown. The indicators "*" and "?" are filled to the defined column length or the column length defined by a current COLUMN command.

OFF displays null values in place of column values for users without authorization, and in place of column values where the security level is unknown.

UNAUTH[ORIZED] *text* enables you to specify the text to be displayed in a secured column for users without authorization. This text appears instead of the default "*****"

You can specify any alpha numeric text up to the column length or a maximum of 30 characters. Longer text is truncated. Text containing spaces must be quoted.

UNK[NOWN] *text* enables you to specify the text to be displayed in a column of unknown security level. This text appears instead of the default "??????"

You can specify any alpha numeric text up to the column length or a maximum of 30 characters. Longer text is truncated. Text containing spaces must be quoted.

Example

```
SET SECUREDCOL ON
SELECT empno, ename, sal FROM emp ORDER BY deptno;


EMPNO ENAME   DEPTNO SAL
 7539 KING    10     ********
 7369 SMITH   20           800
 7566 JONES   20          2975
 7788 SCOTT   20          3000
 7521 WARD    30     ********
 7499 ALLEN   30     ********


SET SECUREDCOL ON UNAUTH notallow
SELECT empno, ename, sal FROM emp ORDER BY deptno;


EMPNO ENAME   DEPTNO SAL
 7539 KING    10     notallow
 7369 SMITH   20           800
 7566 JONES   20          2975
 7788 SCOTT   20          3000
 7521 WARD    30     notallow
 7499 ALLEN   30     notallow
```

13.45.50 SET SERVEROUTPUT

Syntax

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNL[IMITED]}] [FOR[MAT]
{WRA[PPED]  | WOR[D_WRAPPED] | TRU[NCATED]}]
```

Controls whether to display output (that is, DBMS_OUTPUT.PUT_LINE) of stored procedures or PL/SQL blocks in SQL*Plus. The DBMS_OUTPUT line length limit is 32767 bytes.

OFF suppresses the output of DBMS_OUTPUT.PUT_LINE. ON displays the output.

ON uses the SIZE and FORMAT of the previous SET SERVEROUTPUT ON SIZE *n* FORMAT *f*, or uses default values if no SET SERVEROUTPUT command was previously issued in the current connection.

SIZE sets the number of bytes of the output that can be buffered within the Oracle Database server. The default is UNLIMITED. *n* cannot be less than 2000 or greater than 1,000,000.

Resources are not pre-allocated when SERVEROUTPUT is set. As there is no performance penalty, use UNLIMITED unless you want to conserve physical memory.

Every server output line begins on a new output line.

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 detailed information about using UTL_FILE and associated utilities, see the *Oracle Database PL/SQL Packages and Types Reference* .

For more information on DBMS_OUTPUT.PUT_LINE, see Developing Applications with Oracle XA.

Example

To enable text display in a PL/SQL block using DBMS_OUTPUT.PUT_LINE, enter

```
SET SERVEROUTPUT ON
```

The following example shows what happens when you execute an anonymous procedure with SET SERVEROUTPUT ON:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('Task is complete');
END;
/


Task is complete.
PL/SQL procedure successfully completed.
```

The following example shows what happens when you create a trigger with SET SERVEROUTPUT ON:

```
CREATE TABLE SERVER_TAB (Letter CHAR);
CREATE TRIGGER SERVER_TRIG BEFORE INSERT OR UPDATE -
OR DELETE
ON SERVER_TAB
BEGIN
DBMS_OUTPUT.PUT_LINE('Task is complete.');
END;
/


Trigger Created.
```

```
INSERT INTO SERVER_TAB VALUES ('M');
DROP TABLE SERVER_TAB;
/* Remove SERVER_TAB from database */


Task is complete.
1 row created.
```

To set the output to WORD_WRAPPED, enter

```
SET SERVEROUTPUT ON FORMAT WORD_WRAPPED
SET LINESIZE 20
BEGIN
  DBMS_OUTPUT.PUT_LINE('If there is nothing left to do');
  DBMS_OUTPUT.PUT_LINE('shall we continue with plan B?');
END;
/


If there is nothing
left to do
shall we continue
with plan B?
```

To set the output to TRUNCATED, enter

```
SET SERVEROUTPUT ON FORMAT TRUNCATED
SET LINESIZE 20
BEGIN
  DBMS_OUTPUT.PUT_LINE('If there is nothing left to do');
  DBMS_OUTPUT.PUT_LINE('shall we continue with plan B?');
END;
/


If there is nothing
shall we continue wi
```

13.45.51 SET SHIFTINOUT

Syntax

```
SET SHIFT[INOUT] {VIS[IBLE] | INV[ISIBLE]}
```

Enables 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.

Example

To enable the display of shift characters on a terminal that supports them, enter

```
SET SHIFTINOUT VISIBLE
SELECT LAST_NAME, JOB_ID FROM EMP_DETAILS_VIEW
WHERE SALARY > 12000;


LAST_NAME      JOB_ID
---------- ----------
:JJOO:        :AABBCC:
:AA:abc       :DDEE:e
```

where ":" = visible shift character uppercase represents multibyte characters

lowercase represents singlebyte characters

13.45.52 SET SHOWMODE

Syntax

```
SET SHOW[MODE] {ON | OFF}
```

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.

13.45.53 SET SQLBLANKLINES

Syntax

```
SET SQLBL[ANKLINES] {ON | OFF}
```

Controls whether SQL*Plus puts blank lines within a SQL command or script. ON interprets blank lines and new lines as part of a SQL command or script. OFF, the default value, does not allow blank lines or new lines in a SQL command or script or script.

Enter the BLOCKTERMINATOR to stop SQL command entry without running the SQL command. Enter the SQLTERMINATOR character to stop SQL command entry and run the SQL statement.

Example

To allow blank lines in a SQL statement, enter

```
SET SQLBLANKLINES ON
REM Using the SQLTERMINATOR (default is ";")
REM Could have used the BLOCKTERMINATOR (default is ".")
SELECT *

FROM

DUAL

;
```

The following output results:

```
D
-
X
```

13.45.54 SET SQLCASE

Syntax

```
SET SQLC[ASE] {MIX[ED] | 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, to uppercase if SQLCASE equals UPPER, to lowercase if SQLCASE equals LOWER, and makes no changes if SQLCASE equals MIXED.

SQLCASE does not change the SQL buffer itself.

13.45.55 SET SQLCONTINUE

Syntax

```
SET 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 (–).

Example

To set the SQL*Plus command continuation prompt to an exclamation point followed by a space, enter

```
SET SQLCONTINUE '! '
```

SQL*Plus will prompt for continuation as follows:

```
TTITLE 'MONTHLY INCOME' -
! RIGHT SQL.PNO SKIP 2 -
! CENTER 'PC DIVISION'
```

The default continuation prompt is "> ".

13.45.56 SET SQLNUMBER

Syntax

```
SET SQLN[UMBER] {ON | OFF}
```

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.

13.45.57 SET SQLPLUSCOMPATIBILITY

Syntax

```
SET SQLPLUSCOMPAT[IBILITY] {x.y[.z]}
```

Sets the behavior to that of the release or version specified by *x.y*[.*z*].

Where *x* is the version number, *y* is the release number, and *z* is the update number. For example, 8.1.7, 9.0.1 or 10.2. The features affected by SQLPLUSCOMPATIBILITY are tabulated in the SQL*Plus Compatibility Matrix shown. You can also set the value of SQLPLUSCOMPATIBILITY using the -C[OMPATIBILITY] argument of the SQLPLUS command when starting SQL*Plus from the command line.

The default setting for SQLPLUSCOMPATIBILITY is the value of the SQL*Plus client.

It is recommended that you add SET SQLPLUSCOMPATIBILITY 12.2 to your scripts to maximize their compatibility with future versions of SQL*Plus.

13.45.57.1 SQL*Plus Compatibility Matrix

The SQL*Plus Compatibility Matrix tabulates behavior affected by each SQL*Plus compatibility setting. SQL*Plus compatibility modes can be set in three ways:

- You can include a SET SQLPLUSCOMPATIBILITY command in your site or user profile. On installation, there is no SET SQLPLUSCOMPATIBILITY setting in glogin.sql. Therefore the default compatibility is 12.2.

- You can use the SQLPLUS -C[OMPATIBILITY] {*x.y*[.*z*]} command argument at startup to set the compatibility mode of that session.

- You can use the SET SQLPLUSCOMPATIBILITY {*x.y*[.*z*]} command during a session to set the SQL*Plus behavior you want for that session.

The following table shows the release of SQL*Plus which introduced the behavior change, and hence the minimum value of SQLPLUSCOMPATIBILITY to obtain that behavior. For example, to obtain the earlier behavior of the VARIABLE command, you must either use a version of SQL*Plus earlier than 9.0.1, or you must use a SQLPLUSCOMPATIBILITY value of less than 9.0.1. The lowest value that can be set for SQLPLUSCOMPATIBILITY is 7.3.4

Table 13-6   Compatibility Matrix

Value	Consequence	When available
>=10.1	SHOW ERRORS sorts PL/SQL error messages using new columns only available in Oracle Database 10*g*.	10.1
>=10.1	SPOOL Options CREATE, REPLACE, SAVE were added which may affect filename parsing on some platforms.	10.1
>=10.1	SET SQLPROMPT	10.1
>=10.1	Whitespace characters are allowed in Windows file names that are enclosed in quotes. Some other special punctuation characters are now disallowed in Windows.	10.1

Table 13-6   (Cont.) Compatibility Matrix

Value	Consequence	When available
>=10.1	Glogin/login files are called for each reconnect.	10.1
<10.1	Uses the obsolete DOC> prompt when echoing /* comments.	10.1
>= 9.2	A wide column defined FOLD_AFTER may be displayed at the start of a new line. Otherwise it is incorrectly put at the end of the preceding line with a smaller width than expected.	9.2.
>= 9.0	Whitespace before a slash ("/") in a SQL statement is ignored and the slash is taken to mean execute the statement. Otherwise the slash is treated as part of the statement, for example, as a division sign.	9.0.1.4.
>= 9.0	The length specified for NCHAR and NVARCHAR2 types is characters. Otherwise the length may represent bytes or characters depending on the character set.	9.0.1

13.45.58 SET SQLPREFIX

Syntax

```
SET 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.

13.45.59 SET SQLPROMPT

Syntax

```
SET SQLP[ROMPT] {SQL> | text}
```

Sets the SQL*Plus command prompt. SET SQLPROMPT substitute variables dynamically. This enables the inclusion of runtime variables such as the current connection identifier. Substitution variables used in SQLPROMPT do not have to be prefixed with '&', and they can be used and accessed like any other substitution variable. Variable substitution is not attempted for the default prompt, "SQL> ".

Variable substitution occurs each time SQLPROMPT is SET. If SQLPROMPT is included in glogin.sql, then substitution variables in SQLPROMPT are refreshed with each login or connect.

Example

To change your SQL*Plus prompt to display your connection identifier, enter:

```
SET SQLPROMPT "_CONNECT_IDENTIFIER > "
```

To set the SQL*Plus command prompt to show the current user, enter

```
SET SQLPROMPT "_USER > "
```

To change your SQL*Plus prompt to display your the current date, the current user and the users privilege level, enter:

```
SET SQLPROMPT "_DATE _USER _PRIVILEGE> "
```

To change your SQL*Plus prompt to display a variable you have defined, enter:

```
DEFINE mycon = Prod1
SET SQLPROMPT "mycon> "


Prod1>
```

Text in nested quotes is not parsed for substitution. To have a SQL*Plus prompt of your username, followed by "@", and then your connection identifier, enter:

```
SET SQLPROMPT "_USER'@'_CONNECT_IDENTIFIER > "
```

13.45.60 SET SQLTERMINATOR

Syntax

```
SET SQLT[ERMINATOR] {; | c | ON | OFF}
```

Sets the character used to end script or data entry for PL/SQL blocks or SQL statements, to execute the script, and to load it into the buffer.

It cannot be an alphanumeric character or a whitespace. OFF means that SQL*Plus recognizes no command terminator; you terminate a SQL command by entering an empty line or a slash (/). If SQLBLANKLINES is set ON, you must use the BLOCKTERMINATOR to terminate a SQL command. ON resets the terminator to the default semicolon (;).

13.45.61 SET STATEMENTCACHE

Syntax

```
SET STATEMENTC[ACHE] {0 | n}
```

Sets the statement cache size. The cache value allows caching of repeated statements. Therefore these statements do not need to be parsed again, resulting in performance improvement.

Example

To set the statement cache size to 15, enter

```
SET STATEMENTCACHE 15
```

If you do not specify a value for *n*, the default is 0. This means that statement caching is off.

The statement cache size can be any value between 0 and 32767.

> **Note:**
>
> Specify a cache size that is less than the value specified for the open cursor parameter in the Oracle Database initialization file, init.ora. Specifying a value of 0 will switch off statement caching. The `<statement_cache>` setting in the `oraaccess.xml` file can override the SET STATEMENTCACHE setting in SQL*Plus. For more information about `oraaccess.xml`, see the *Oracle Call Interface Programmer's Guide*.

To show the current setting for the statement cache size, enter

```
SHOW STATEMENTC[ACHE]
```

13.45.62 SET SUFFIX

Syntax

```
SET SUF[FIX] {SQL | text}
```

Sets the default file extension that SQL*Plus uses in commands that refer to scripts. SUFFIX does not control extensions for spool files.

Example

To change the default command-file extension from the default, .SQL to .TXT, enter

```
SET SUFFIX TXT
```

If you then enter

```
GET EXAMPLE
```

SQL*Plus will look for a file named EXAMPLE.TXT instead of EXAMPLE.SQL.

13.45.63 SET TAB

Syntax

```
SET TAB {ON | OFF}
```

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.

13.45.64 SET TERMOUT

Syntax

```
SET TERM[OUT] {ON | OFF}
```

Controls the display of output generated by commands in a script that is executed with @, @@ or START. OFF suppresses the display so that you can spool output to a file without displaying the output on screen. ON displays the output on screen. TERMOUT OFF does not affect output from commands you enter interactively or redirect to SQL*Plus from the operating system.

13.45.65 SET TIME

Syntax

```
SET TI[ME] {ON | OFF}
```

Controls the display of the current time. ON displays the current time before each command prompt. OFF suppresses the time display.

13.45.66 SET TIMING

Syntax

```
SET TIMI[NG] {ON | OFF}
```

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.

See TIMING for information on timing multiple commands.

Example

The format of timing statistics is dependent on the operating system. In Linux and Windows, the timing statistics are in 24 hour format displaying hours, minutes, seconds and hundredths of seconds

```
SET SUFFIX TXT
```

If you enter

```
GET EXAMPLE
```

SQL*Plus displays output like

```
GET EXAMPLE
```

13.45.67 SET TRIMOUT

Syntax

```
SET TRIM[OUT] {ON | OFF}
```

Determines whether SQL*Plus puts 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 enables SQL*Plus to display trailing blanks. TRIMOUT ON does not affect spooled output.

13.45.68 SET TRIMSPOOL

Syntax

```
SET TRIMS[POOL] {ON | OFF}
```

Determines whether SQL*Plus puts trailing blanks at the end of each spooled line. ON removes blanks at the end of each line. OFF enables SQL*Plus to include trailing blanks. TRIMSPOOL ON does not affect terminal output.

13.45.69 SET UNDERLINE

Syntax

```
SET UND[ERLINE] {- | c | ON | OFF}
```

Sets the character used to underline column headings in reports. The underline character 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 "-".

13.45.70 SET VERIFY

Syntax

```
SET VER[IFY] {ON | OFF}
```

Controls whether to list the text of a SQL statement or PL/SQL command before and after replacing substitution variables with values. ON lists the text; OFF suppresses the listing.

13.45.71 SET WRAP

Syntax

```
SET WRA[P] {ON | OFF}
```

Controls whether to truncate the display of a selected row if it is too long for the current line width. OFF truncates the selected row; ON enables 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.

13.45.72 SET XMLOPTIMIZATIONCHECK

Syntax

```
SET XMLOPT[IMIZATIONCHECK] [ON|OFF]
```

Controls whether only XML queries and DML operations that are fully optimized are executed. ON prevents the execution of any XML query or DML operation that cannot be fully optimized and writes the reason in the trace file. OFF does not prevent the execution of such queries and operations. OFF is the default.

SET XMLOPT[IMIZATIONCHECK] ON is only to assist during development and debugging an XML query or DML operation.

13.45.73 SET XQUERY BASEURI

Syntax

```
SET XQUERY BASEURI {text}
```

Specifies the base URI used to resolve relative URIs in functions. It enables the prefix of the file accessed by an XQuery to be changed.

To unset the BASEURI, set an empty string, for example:

```
SET XQUERY BASEURI ''
```

Take care to enter valid values as values are checked only when an XQUERY command is issued.

Example

```
SET XQUERY BASEURI '/public/scott'
XQUERY for $i in doc("foo.xml") return $i
/
```

This is equivalent to:

```
XQuery declare base-uri "/public/hr";
for $i in doc("foo.xml") return $i
```

13.45.74 SET XQUERY ORDERING

Syntax

```
SET XQUERY ORDERING {UNORDERED | ORDERED | DEFAULT}
```

Sets the ordering of output from an XQuery. There are three values:

UNORDERED specifies that results are sorted in the order they are retrieved from the database.

ORDERED specifies that results are sorted as defined by the XQuery.

DEFAULT specifies the database default. In Oracle Database 10*g* the default is
UNORDERED.

When SET XQUERY ORDERING is not set, the default is DEFAULT (UNORDERED).

Example

```
SET XQUERY ORDERING ORDERED
XQUERY for $i in doc("foo.xml") return $i
/
```

This is equivalent to:

```
XQuery declare ordering ordered;
for $i in doc("foo.xml") return $i
/
```

13.45.75 SET XQUERY NODE

Syntax

```
SET XQUERY NODE {BYVALUE | BYREFERENCE | DEFAULT}
```

Sets the node identity preservation mode. The preservation mode applies to all
expressions that either create a new node (such as element constructors) or return an
item or sequence containing nodes (such as path expressions). There are three
values:

BYVALUE specifies that the node identity need not be preserved. This means that any
node operation such as creation, or that is returned as a result of an expression is
deep copied and loses its context from the original tree. Subsequent operations on this
node that test for node identity, parent or sibling axes or ordering will be undefined.

BYREFERENCE specifies that node identities are to be preserved. Subsequent
operations on this node preserve the node's context and definition.

DEFAULT specifies the database default. In Oracle Database 10*g* the default is
BYVALUE.

When SET XQUERY NODE is not set, the default is DEFAULT (BYVALUE).

Example

```
SET XQUERY NODE BYREFERENCE
XQUERY for $i in doc("foo.xml") return $i
/
```

This is equivalent to:

```
XQuery declare node byreference;
for $i in doc("foo.xml") return $i
/
```

13.45.76 SET XQUERY CONTEXT

Syntax

```
SET XQUERY CONTEXT {text}
```

Specifies an XQuery context item expression. A context item expression evaluates to the context item, which may be either a node (as in the expression fn:doc("bib.xml")//book[fn:count(./author)>1]), or an atomic value (as in the expression (1 to 100)[. mod 5 eq 0]).

To unset the XQUERY CONTEXT, set an empty string, for example:

```
SET XQUERY CONTEXT ''
```

Take care to enter valid values as values are checked only when an XQUERY command is issued.

Example

```
SET XQUERY CONTEXT 'doc("foo.xml")'
XQUERY for $i in /a return $i
/
```

This is equivalent to:

```
XQuery for $i in /a return $i
passing XMLQuery("doc('foo.xml')")
/
```

13.46 SHOW

Syntax

```
SHO[W] option
```

where *option* represents one of the following terms or clauses:

```
system_variable  ALL BTI[TLE]   CON_ID  CON_NAME EDITION   ERR[ORS] [ {ANALYTIC VIEW |
ATTRIBUTE DIMENSION | HIERARCHY | FUNCTION | PROCEDURE | PACKAGE | PACKAGE BODY |
TRIGGER | VIEW | TYPE | TYPE BODY | DIMENSION | JAVA CLASS } [schema.]name]HISTORY
LNO  LOBPREF[ETCH]   PARAMETER[S] [parameter_name]  PDBS PNO  RECYC[LEBIN]
[original_name]  REL[EASE]  REPF[OOTER]  REPH[EADER]  ROWPREF[ETCH] SGA SPOO[L]
SPPARAMETER[S] [parameter_name]  SQLCODE STATEMENTC[ACHE] TTI[TLE] USER XQUERY
```

Shows the value of a SQL*Plus system variable or the current SQL*Plus environment. SHOW SGA requires a DBA privileged login.

Terms

```
system_variable
```

Represents any system variable set by the SET command.

```
ALL
```

Lists the settings of all SHOW options, except ERRORS and SGA, in alphabetical order.

```
CON_ID
```

Displays the id of the Container to which you are connected when connected to a Consolidated Database.

```
CON_NAME
```

Displays the name of the Container to which you are connected when connected to a Consolidated Database.

```
EDITION
```

Shows the edition attribute of the existing database.

```
BTI[TLE]
```

Shows the current BTITLE definition.

```
ERR[ORS] [{ANALYTIC VIEW | ATTRIBUTE DIMENSION | HIERARCHY | FUNCTION |
PROCEDURE | PACKAGE | PACKAGE BODY | TRIGGER  | VIEW | TYPE | TYPE BODY |
DIMENSION | JAVA CLASS} [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 (analytic view, attribute dimension, hierarchy, function, procedure, package, package body, trigger, view, type, type body, dimension, or java class) 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.

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 use the COLUMN command to alter the default widths.

```
HISTORY
```

Shows the current command history status that is set by using the SET HISTORY command.

```
LNO
```

Shows the current line number (the position in the current page of the display and/or spooled output).

```
LOBPREFETCH
```

Shows the current setting for the amount of LOB data that SQL*Plus will prefetch from the database at one time. For more information about setting the amount of LOB data that SQL*Plus will prefetch from the database at one time, see SET LOBPREFETCH.

```
PARAMETERS [parameter_name]
```

Displays the current values for one or more initialization parameters. You can use a string after the command to see a subset of parameters whose names include that string. For example, if you enter:

```
SHOW PARAMETERS COUNT


NAME                             TYPE      VALUE
-----------------------------    -----     -----
db_file_multiblock_read_count    integer   12
spin_count                       integer   0
```

The SHOW PARAMETERS command, without any string following the command, displays all initialization parameters.

Your output may vary depending on the version and configuration of the Oracle Database server to which you are connected. You need SELECT ON V_$PARAMETER object privileges to use the PARAMETERS clause, otherwise you will receive a message

```
ORA-00942: table or view does not exist
```

```
PDBS
```

Display the names, ids, mode and restriction status of Pluggable Databases in the Consolidated Database to which you are connected.

The PDBS option is only available when you are logged in as SYSDBA and have the SYSDBA privilege. For non-DBA users, attempting to use the PDBS option returns the error SP2-0382: The SHOW PDBS command is not available.

```
PNO
```

Shows the current page number.

```
RECYC[LEBIN] [original_name]
```

Shows objects in the recycle bin that can be reverted with the FLASHBACK BEFORE DROP command. You do not need to remember column names, or interpret the less readable output from the query:

```
SELECT * FROM USER_RECYCLEBIN
```

The query returns four columns displayed in the following order:

Column Name	Description
ORIGINAL NAME	Shows the original name used when creating the object.
RECYCLEBIN NAME	Shows the name used to identify the object in the recyclebin.
OBJECT TYPE	Shows the type of the object.
DROP TIME	Shows the time when the object was dropped.

The output columns can be formatted with the COLUMN command.

For DBAs, the command lists their own objects as they have their own user_recyclebin view.

```
REL[EASE]
```

Shows the release number of Oracle Database that SQL*Plus is accessing.

```
REPF[OOTER]
```

Shows the current REPFOOTER definition.

```
REPH[EADER]
```

Shows the current REPHEADER definition.

```
ROWPREFETCH
```

Shows the current setting for the number of rows that SQL*Plus will prefetch from the database at one time. For more information about setting the number of rows that SQL*Plus will prefetch from the database at one time, see SET ROWPREFETCH.

```
SPOO[L]
```

Shows whether output is being spooled.

```
SGA
```

Displays information about the current instance's System Global Area. You need SELECT ON V_$SGA object privileges otherwise you will receive a message

```
ORA-00942: table or view does not exist
```

```
SPPARAMETERS [parameter_name]
```

As for SHOW PARAMETERS except that SHOW SPPARAMETERS displays current values for initialization parameters for all instances. You can use a string after the command to see a subset of parameters whose names include that string.

The SHOW SPPARAMETERS command, without any string following the command, displays all initialization parameters for all instances.

Your output may vary depending on the version and configuration of the Oracle Database server to which you are connected. You need SELECT ON V_$PARAMETER object privileges to use the SPPARAMETERS clause.

```
SQLCODE
```

Shows the value of SQL.SQLCODE (the SQL return code of the most recent operation).

```
STATEMENTCACHE
```

Shows the current setting for the statement cache size. For more information about setting the statement cache size, see SET STATEMENTCACHE.

```
TTI[TLE]
```

Shows the current TTITLE definition.

```
USER
```

Shows the username you are currently using to access SQL*Plus. If you connect as "/ AS SYSDBA", then the SHOW USER command displays

```
USER is "SYS"
```

```
XQUERY
```

Shows the current values of the XQUERY settings, BASEURI, CONTEXT, NODE and ORDERING.

```
xquery BASEURI "public/scott" CONTEXT "doc("test.xml")" NODE byreference ORDERING
ordered
```

The following output is displayed when no values are set:

```
xquery BASEURI "" CONTEXT "" NODE default ORDERING default
```

Examples

To display information about the SGA, enter

```
SHOW SGA


Total System Global Area                                7629732 bytes
Fixed Size                                                60324 bytes
Variable Size                                           6627328 bytes
Database Buffers                                         409600 bytes
Redo Buffers                                             532480 bytes
```

The following example illustrates how to create a stored procedure and then show its compilation errors:

```
CONNECT SYSTEM/MANAGER
CREATE PROCEDURE HR.PROC1 AS
BEGIN
:P1 := 1;
END;
/


Warning: Procedure created with compilation errors.


SHOW ERRORS PROCEDURE PROC1


NO ERRORS.


SHOW ERRORS PROCEDURE HR.PROC1


Errors for PROCEDURE HR PROC1:
LINE/COL ERROR
---------------------------------------------------------
3/3     PLS-00049: bad bind variable 'P1'
```

To show whether AUTORECOVERY is enabled, enter

```
SHOW AUTORECOVERY


AUTORECOVERY ON
```

To display the id of the container to which you are connected, enter

```
SHOW CON_ID

CON_ID
------------------------------
1
```

To display the name of the container to which you are connected, enter

```
SHOW CON_NAME

CON_NAME
------------------------------
CDB$ROOT
```

To display the current command history status that is set by issuing the SET
HIST[ORY] {ON | OFF | n} command, enter

```
SHOW HISTORY

SQL> set history on
SQL> show history
History is ON and set to "100"
SQL> set history off
SQL> show history
History is OFF
SQL> set history 1000
SQL> show history
History is ON and set to "1000"
```

To display the names, ids, and modes of Pluggable Databases in the Consolidated
Database to which you are connected, enter

```
SHOW PDBS

CON_ID CON_NAME    OPEN MODE     RESTRICTED
------ ---------- ------------ -----------
2      PDB$SEED    READ ONLY     NO
3      CDB1_PDB1   READ WRITE    NO
```

To display the connect identifier for the default instance, enter

```
SHOW INSTANCE

INSTANCE "LOCAL"
```

To display the location for archive logs, enter

```
SHOW LOGSOURCE

LOGSOURCE "/usr/oracle90/dbs/arch"
```

To display objects that can be reverted with the FLASHBACK commands where CJ1 and ABC were objects dropped, enter:

```
SHOW RECYCLEBIN


ORIGINAL NAME       RECYCLEBIN NAME     OBJECT TYPE     DROP TIME
--------------      ------------------  ------------    --------------------
CJ1                 RB$$29458$TABLE$0   TABLE           2003-01-22:14:54:07
ABC                 RB$$29453$TABLE$0   TABLE           2003-01-20:18:50:29
```

To restore CJ1, enter

```
FLASHBACK TABLE CJ1 TO BEFORE DROP;
```

13.47 SHOW CONNECTION

Syntax

```
SHOW CONN[ECTION] NETS[ERVICENAMES] [net_service_name_1 net_service_name_2 ..
net_service_name_n]
```

This command lists the Oracle Net service names present in the `tnsnames.ora` file and resolves the given Oracle Net service name to the connection string.

When the `SHOW CONNECTION` command is run with one or more Oracle Net service names, it displays the corresponding connection strings.

Terms

```
SHOW CONN[ECTION] NETS[ERVICENAMES]
```

Lists all the Oracle Net service names present in the `tnsnames.ora` file.

```
net_service_name
```

A net service name. You can also provide an Easy Connect string in place of the Oracle Net service name. When the `SHOW CONNECTION` command is executed with this value, it displays a connection string for the Oracle Net service name or an Easy Connect string.

Examples

You can view the Oracle Net services names from the `tnsnames.ora` file by executing the following command:

```
SHOW CONNECTION NETSERVICENAMES
```

The Oracle Net services names from the `tnsnames.ora` file are displayed:

```
Local Net Naming configuration file: /opt/oracle/tnsnames.ora
cdb1_pdb1
cdb1_pdb2
```

You can view details about an Oracle Net service name by executing the following command:

```
SHOW CONNECTION NETSERVICENAMES cdb1_pdb1
```

The details of an Oracle Net service name are displayed:

```
Local Net Naming configuration file: /opt/oracle/tnsnames.ora
cdb1_pdb1 = (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=cdb1_pdb1))
(ADDRESS=(PROTOCOL=tcp)(HOST=host1.customer.domain.com)(PORT=1521)))
```

13.48 SHUTDOWN

Syntax

```
SHUTDOWN [ABORT | IMMEDIATE | NORMAL | TRANSACTIONAL [LOCAL]]
```

Shuts down a currently running Oracle Database instance, optionally closing and dismounting a database. If the current database is a pluggable database, only the pluggable database is closed. The consolidated instance continues to run.

Shutdown commands that wait for current calls to complete or users to disconnect such as SHUTDOWN NORMAL and SHUTDOWN TRANSACTIONAL have a time limit that the SHUTDOWN command will wait. If all events blocking the shutdown have not occurred within the time limit, the shutdown command cancels with the following message:

```
ORA-01013: user requested cancel of current operation
```

Prerequisites for PDB Shutdown

When the current container is a pluggable database (PDB), the SHUTDOWN command can only be used if:

- The current user has SYSDBA, SYSOPER, SYSBACKUP, or SYSDG system privilege.
- The privilege is either commonly granted or locally granted in the PDB.
- The current user exercises the privilege using AS SYSDBA, AS SYSOPER, AS SYSBACKUP, or AS SYSDG at connect time.
- To close a PDB, the PDB must be open.

For more information, see the *Oracle Database Administrator's Guide*

Terms

```
ABORT
```

Proceeds with the fastest possible shutdown of the database without waiting for calls to complete or users to disconnect.

Uncommitted transactions are not rolled back. Client SQL statements currently being processed are terminated. All users currently connected to the database are implicitly disconnected and the next database startup will require instance recovery.

You must use this option if a background process terminates unusually.

```
IMMEDIATE
```

Does not wait for current calls to complete or users to disconnect from the database.

Further connects are prohibited. The database is closed and dismounted. The instance is shutdown and no instance recovery is required on the next database startup.

```
NORMAL
```

NORMAL is the default option which waits for users to disconnect from the database.

Further connects are prohibited. The database is closed and dismounted. The instance is shutdown and no instance recovery is required on the next database startup.

```
TRANSACTIONAL [LOCAL]
```

Performs a planned shutdown of an instance while allowing active transactions to complete first. It prevents clients from losing work without requiring all users to log off.

No client can start a new transaction on this instance. Attempting to start a new transaction results in disconnection. After completion of all transactions, any client still connected to the instance is disconnected. Now the instance shuts down just as it would if a SHUTDOWN IMMEDIATE statement was submitted. The next startup of the database will not require any instance recovery procedures.

The LOCAL mode specifies a transactional shutdown on the local instance only, so that it only waits on local transactions to complete, not all transactions. This is useful, for example, for scheduled outage maintenance.

Usage

SHUTDOWN with no arguments is equivalent to SHUTDOWN NORMAL.

You must be connected to a database as SYSDBA, SYSOPER, SYSBACKUP, or SYSDG. You cannot connect through a multi-threaded server. See CONNECT for more information about connecting to a database.

Examples

To shutdown a CDB, you must be connected to the CDB instance that you want to close, and then enter

```
SHUTDOWN

Database closed.
Database dismounted.
Oracle instance shut down.
```

To shutdown a PDB, you must log into the PDB to issue the SHUTDOWN command.

```
SHUTDOWN

Pluggable Database closed.
```

13.49 SPOOL

Syntax

```
SPO[OL] [file_name[.ext] [CRE[ATE] | REP[LACE] | APP[END]] | OFF | OUT]
```

Stores query results in a file, or optionally sends the file to a printer.

Terms

```
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). The extension is not appended to system files such as /dev/null and /dev/stderr.

```
CRE[ATE]
```

Creates a new file with the name specified.

```
REP[LACE]
```

Replaces the contents of an existing file. If the file does not exist, REPLACE creates the file. This is the default behavior.

```
APP[END]
```

Adds the contents of the buffer to the end of the file you specify.

```
OFF
```

Stops spooling.

```
OUT
```

Stops spooling and sends the file to your computer's standard (default) printer. This option is not available on some operating systems.

Enter SPOOL with no clauses to list the current spooling status.

Usage

To spool output generated by commands in a script without displaying the output on the screen, use SET TERMOUT OFF. SET TERMOUT OFF does not affect output from commands that run interactively.

You must use quotes around file names containing white space.

To create a valid HTML file using SPOOL APPEND commands, you must use PROMPT or a similar command to create the HTML page header and footer. The SPOOL APPEND command does not parse HTML tags.

SET SQLPLUSCOMPAT[IBILITY] to 9.2 or earlier to disable the CREATE, APPEND and SAVE parameters. See **SQL*Plus Compatibility Matrix** to determine what functionality is controlled by the SET SQLPLUSCOMPAT[IBILITY] command.

Examples of SPOOL Command

To record your output in the new file DIARY using the default file extension, enter

```
SPOOL DIARY CREATE
```

To append your output to the existing file DIARY, enter

```
SPOOL DIARY APPEND
```

To record your output to the file DIARY, overwriting the existing content, enter

```
SPOOL DIARY REPLACE
```

To stop spooling and print the file on your default printer, enter

```
SPOOL OUT
```

13.50 START

Syntax

```
STA[RT] {url | file_name[.ext] } [arg...]
```

Runs the SQL*Plus statements in the specified script. The script can be called from the local file system or from a web server.

Terms

url

Specifies the Uniform Resource Locator of a script to run on the specified web server. SQL*Plus supports HTTP and FTP protocols, but not HTTPS. HTTP authentication in the form http://*username:password@machine_name.domain*... is not supported in this release.

file_name[*.ext*]

The script 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). See SET SUF[FIX] {SQL | text} for information on changing the default extension.

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. See the platform-specific Oracle documentation provided for your operating system for specific information related to your operating system environment.

arg ...

Data items you wish to pass to parameters in the script. If you enter one or more arguments, SQL*Plus substitutes the values into the parameters (&1, &2, and so forth) in the script. The first argument replaces each occurrence of &1, the second replaces each occurrence of &2, and so on.

The START command defines the parameters with the values of the arguments; if you START the script again in this session, you can enter new arguments or omit the arguments to use the old values.

See **Defining Substitution Variables** and **Using Substitution Variables** for more information on using parameters.

Usage

All previous settings like COLUMN command settings stay in effect when the script starts. If the script changes any setting, then this new value stays in effect after the script has finished

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 **@ (at sign)** and **@@ (double at sign)** for further information on these commands. See **Disabling SQL*Plus_ SQL_ and PL/SQL Commands** for more information.

The EXIT or QUIT command in a script terminates SQL*Plus.

Examples

A file named PROMOTE with the extension SQL, used to promote employees, might contain the following command:

```
SELECT FIRST_NAME, LAST_NAME, JOB_ID, SALARYFROM EMP_DETAILS_VIEWWHERE
JOB_ID='&1' AND SALARY>&2;
```

To run this script, enter

```
START PROMOTE ST_MAN 7000
```

or if it is located on a web server, enter a command in the form:

```
START HTTP://machine_name.domain:port/PROMOTE.SQL ST_MAN 7000
```

Where *machine_name.domain* must be replaced by the host.domain name, and *port* by the port number used by the web server where the script is located.

The following command is executed:

```
SELECT LAST_NAME, LAST_NAME
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='ST_MAN' AND SALARY>7000;
```

and the results displayed.

13.51 STARTUP

Syntax

```
STARTUP db_options  | cdb_options | upgrade_options
```

where *db options* has the following syntax:

```
[FORCE] [RESTRICT] [PFILE=filename] [QUIET]  [ MOUNT [dbname] |  [ OPEN
[open_db_options] [dbname] ] | NOMOUNT ]
```

where *open_db_options* has the following syntax:

```
READ {ONLY | WRITE [RECOVER]} | RECOVER
```

where *cdb_options* has the following syntax:

```
root_connection_options | pdb_connection_options
```

where *root_connection_options* has the following syntax:

```
PLUGGABLE DATABASE pdbname  [FORCE] | [UPGRADE] | [RESTRICT] [ OPEN {open_pdb_options}]
```

where *pdb_connection_options* has the following syntax:

```
[FORCE] | [UPGRADE] | [RESTRICT] [ OPEN {open_pdb_options}]
```

where *open_pdb_options* has the following syntax:

```
READ WRITE | READ ONLY
```

and where *upgrade_options* has the following syntax:

```
[PFILE=filename] {UPGRADE | DOWNGRADE} [QUIET]
```

Starts an Oracle Database instance with several options, including mounting and opening a database.

Prerequisites for a PDB STARTUP

When the current container is a pluggable database (PDB), the STARTUP command can only be used if:

- The current user has SYSDBA, SYSOPER, SYSBACKUP, or SYSDG system privilege.
- The privilege is either commonly granted or locally granted in the PDB.
- The current user exercises the privilege using AS SYSDBA, AS SYSOPER, SYSBACKUP, or AS SYSDG at connect time.
- The PDB is in MOUNTED mode, excluding the use of the FORCE option.
- The PDB must be in READ ONLY or READ WRITE mode to be in mounted mode.

For more information, see the *Oracle Database Administrator's Guide*

> **Note:**
>
> Only use *db_options* to start a Database.
>
> Only use *root_connection_options* to start a Pluggable Database while connected to the Root.
>
> Only use *pdb_options* to start a Pluggable Database to which you are connected.
>
> Only use *upgrade_options* to start a Database for upgrade or downgrade.
>
> If neither `READ WRITE` nor `READ ONLY` is specified, a PDB will be opened in `READ ONLY` if a CDB to which it belongs is used as a physical standby database, otherwise the PDB will be opened `READ WRITE`.
>
> For more information about using Consolidated and Pluggable Databases, see Creating and Configuring an Oracle Database.

Terms

`FORCE`

Shuts down the current Oracle Database instance (if it is running) with SHUTDOWN mode ABORT, before restarting it. If the current instance is running and FORCE is not specified, an error results. FORCE is useful while debugging and under unusual circumstances. It should not normally be used.

`RESTRICT`

Only enables Oracle Database users with the RESTRICTED SESSION system privilege to connect to the database. Later, you can use the ALTER SYSTEM command to disable the restricted session feature.

`PFILE=`*`filename`*

Specifies the client parameter file to be used while starting the instance. If PFILE is not specified, the server attempts to access a default server parameter file (spfile). If the default spfile isn't found, the server then attempts to access a default pfile. The default files are platform specific. For example, the default file is $ORACLE_HOME/dbs/init$ORACLE_SID.ora on UNIX, and ORACLE_HOME\database\initORCL.ora on Windows.

`QUIET`

Suppresses the display of System Global Area information for the starting instance.

`MOUNT` *`dbname`*

Mounts a database but does not open it.

dbname is the name of the database to mount or open. If no database name is specified, the database name is taken from the initialization parameter DB_NAME.

`OPEN`

Mounts and opens the specified database.

`NOMOUNT`

Causes the database not to be mounted upon instance startup.

Cannot be used with MOUNT, or OPEN.

`RECOVER`

Specifies that media recovery should be performed, if necessary, before starting the instance. STARTUP RECOVER has the same effect as issuing the RECOVER DATABASE command and starting an instance. Only complete recovery is possible with the RECOVER option.

Recovery proceeds, if necessary, as if AUTORECOVERY is set to ON, regardless of whether or not AUTORECOVERY is enabled. If a redo log file is not found in the expected location, recovery continues as if AUTORECOVERY is disabled, by prompting you with the suggested location and name of the subsequent log files that need to be applied.

`UPGRADE`

Starts the database in OPEN UPGRADE mode and sets system initialization parameters to specific values required to enable database upgrade scripts to be run. UPGRADE should only be used when a database is first started with a new version of the Oracle Database Server.

When run, upgrade scripts transform an installed version or release of an Oracle database into a later version, for example, to upgrade an Oracle9*i* database to Oracle Database 10*g*. Once the upgrade completes, the database should be shut down and restarted normally.

```
DOWNGRADE
```

Starts the database in OPEN DOWNGRADE mode and sets system initialization parameters to specific values required to enable database downgrade scripts to be run.

When run, downgrade scripts transform an installed version or release of Oracle Database into a previous version, for example, to downgrade an Oracle10*g* database to an Oracle9*i* database. Once the downgrade completes, the database should be shut down and restarted normally.

```
PLUGGABLE DATABASE
```

Use the pluggable database *pdbname* option to specify the plugggable database on which you want the STARTUP command to act.

Usage

You must be connected to a database as SYSDBA, SYSOPER, SYSBACKUP, or SYSDG. You cannot be connected to a shared server via a dispatcher.

STARTUP with no arguments is equivalent to STARTUP OPEN.

STARTUP OPEN RECOVER mounts and opens the database even when recovery fails.

Examples

To start a CDB instance using the standard parameter file, mount the default database, and open the database, enter

```
STARTUP
```

or enter

```
STARTUP OPEN database
```

To start an instance using the standard parameter file, mount the default database, and open the database, enter

```
STARTUP FORCE RESTRICT MOUNT
```

To start an instance using the parameter file TESTPARM without mounting the database, enter

```
STARTUP PFILE=testparm NOMOUNT
```

To shutdown a particular database, immediately restart and open it, allow access only to users with the RESTRICTED SESSION privilege, and use the parameter file MYINIT.ORA. enter

```
STARTUP FORCE RESTRICT PFILE=myinit.ora OPEN database
```

To startup an instance and mount but not open a database, enter

```
CONNECT / as SYSDBA


Connected to an idle instance.


STARTUP MOUNT


ORACLE instance started.

Total System Global Area                         7629732 bytes
Fixed Size                                         60324 bytes
Variable Size                                     6627328 bytes
Database Buffers                                   409600 bytes
Redo Buffers                                       532480 bytes
```

To startup a PDB from a PDB container, enter the following sequence

```
CONNECT SYS/<password>@CDB1_PDB1 AS SYSDBA


Connected.


SHOW CON_NAME


CON_NAME
------------------------------
CDB1_PDB1


SHOW PDBS


    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         3 CDB1_PDB1                      MOUNTED


STARTUP


Pluggable Database opened.


SHOW PDBS


    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         3 CDB1_PDB1                      READ WRITE NO
```

To startup a PDB from root, enter the following sequence

```
CONNECT / AS SYSDBA


Connected.
```

```
SHOW CON_NAME

CON_NAME
------------------------------
CDB$ROOT

SHOW PDBS

    CON_ID CON_NAME                         OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                        READ ONLY  NO
         3 CDB1_PDB1                       MOUNTED

STARTUP PLUGGABLE DATABASE CDB1_PDB1

Pluggable Database opened.

SHOW PDBS

    CON_ID CON_NAME      OPEN MODE  RESTRICTED
---------- ------------- ---------- ----------
         2 PDB$SEED      READ ONLY  NO
         3 CDB1_PDB1     READ WRITE NO
```

13.52 STORE

Syntax

```
STORE SET file_name[.ext] [ CRE[ATE] | REP[LACE] | APP[END]]
```

Saves attributes of the current SQL*Plus environment in a script.

Terms

See SAVE for information on the other terms and clauses in the STORE command syntax.

```
SET
```

Saves the values of the system variables.

Usage

This command creates a script which can be executed with the START, @ (at sign) or @@ (double at sign) 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.

Examples

To store the current SQL*Plus system variables in a file named DEFAULTENV with the default command-file extension, enter

```
STORE SET DEFAULTENV
```

To append the current SQL*Plus system variables to an existing file called
DEFAULTENV with the extension OLD, enter

```
STORE SET DEFAULTENV.OLD APPEND
```

13.53 TIMING

Syntax

```
TIMI[NG] [START text | SHOW | STOP]
```

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.

Terms

```
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. For other information
about TIMING, see SET AUTOTRACE

Usage

You can use this data to do a performance analysis on any commands or blocks run
during the period.

See the SET TIMING command 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

```
TIMING START SQL_TIMER
```

To list the current timer's title and accumulated time, enter

```
TIMING SHOW
```

To list the current timer's title and accumulated time and to remove the timer, enter

```
TIMING STOP
```

13.54 TTITLE

Syntax

```
TTI[TLE] [printspec [text | variable] ...] [ON | OFF]
```

where *printspec* represents one or more of the following clauses used to place and format the text:

```
BOLD CE[NTER] COL n FORMAT text LE[FT] R[IGHT] S[KIP] [n] TAB n
```

Places and formats a specified title at the top of each report page. Enter TTITLE with no clauses to list its current definition. The old form of TTITLE is used if only a single word or string in quotes follows the TTITLE command.

See TTI[TLE] text (obsolete old form) for a description of the old form of TTITLE.

Terms

These terms and clauses also apply to the BTITLE command.

```
text
```

The title text. Enter text in single quotes if you want to place more than one word on a single line.

```
variable
```

A substitution variable or any of the following system-maintained values, SQL.LNO (the current line number), SQL.PNO (the current page number), SQL.RELEASE (the current Oracle Database release number), SQL.SQLCODE (the current error code), or SQL.USER (the current username).

To print one of these values, reference the appropriate variable in the title. You can format *variable* with the FORMAT clause.

SQL*Plus substitution variables (& variables) are expanded before TTITLE is executed. The resulting string is stored as the TTITLE text. During subsequent execution for each page of results, the expanded value of a variable may itself be interpreted as a substitution variable with unexpected results.

You can avoid this double substitution in a TTITLE command by not using the & prefix for variables that are to be substituted on each page of results. If you want to use a substitution variable to insert unchanging text in a TTITLE, enclose it in quotes so that it is only substituted once.

```
OFF
```

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). Here "column" 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] | 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 bold 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 the COLUMN command 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 using 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.

Enter TTITLE with no clauses to list the current TTITLE definition.

Usage

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 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 them 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:

```
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

```
TTITLE LEFT 'Monthly Analysis' CENTER '01 Jan 2003' -
RIGHT 'Page:' FORMAT 999 SQL.PNO SKIP CENTER -
'Data in Thousands'


Monthly Analysis                    01 Jan 2003                         Page: 1
                                  Data in Thousands
```

To suppress the top title display without changing its definition, enter

```
TTITLE OFF
```

13.55 UNDEFINE

Syntax

```
UNDEF[INE] variable ...
```

where *variable* represents the name of the substitution variable you want to delete.

Deletes one or more substitution variables that you defined either explicitly (with the DEFINE command) or implicitly (with an argument to the START command).

Examples

To undefine a substitution variable named POS, enter

```
UNDEFINE POS
```

To undefine two substitution variables named MYVAR1 and MYVAR2, enter

```
UNDEFINE MYVAR1 MYVAR2
```

13.56 VARIABLE

Syntax

```
VAR[IABLE] [variable [type [=value]]]
```

Declares a bind variable that can be referenced in PL/SQL, or lists the current display characteristics for a single variable or all variables.

type represents data types. These data types are listed in the Terms section.

`VARIABLE` without arguments displays a list of all the variables declared in the session. `VARIABLE` followed only by a variable name lists that variable.

To free resources used by `CLOB` and `NCLOB` bind variables, you may need to manually free temporary LOBs with:

```
EXECUTE DBMS_LOB.FREETEMPORARY(:cv)
```

See About Using Bind Variables for more information on bind variables. See your *Oracle Database PL/SQL Language Reference* for more information about PL/SQL.

Terms

variable

Represents the name of the bind variable you wish to create.

value

Allows you to assign a value to a variable for input binding.

NUMBER

Creates a variable of type NUMBER with fixed length.

CHAR

Creates a variable of type CHAR (character) with length one.

CHAR (*n*[CHAR | BYTE])

Creates a variable of type CHAR with length *n* bytes or *n* characters. The maximum that *n* can be is 2000 bytes, and the minimum is 1 byte or 1 character. The maximum *n* for a CHAR variable with character semantics is determined by the number of bytes required to store each character for the chosen character set, with an upper limit of 2000 bytes. The length semantics are determined by the length qualifiers CHAR or BYTE, and if not explicitly stated, the value of the NLS_LENGTH_SEMANTICS environment variable is applied to the bind variable. Explicitly stating the length semantics at variable definition stage will always take precedence over the NLS_LENGTH_SEMANTICS setting.

NCHAR

Creates a variable of type NCHAR (national character) with length one.

NCHAR (*n*)

Creates a variable of type NCHAR with length *n* characters. The maximum that *n* can be is determined by the number of bytes required to store each character for the chosen national character set, with an upper limit of 2000 bytes. The only exception to this is when a SQL*Plus session is connected to a pre Oracle9*i* server, or the SQLPLUSCOMPATIBILITY system variable is set to a version less than 9.0.0. In this case, the length *n* can be in bytes or characters depending on the chosen national character set, with the upper limit of 2000 bytes still retained.

VARCHAR2 (*n*[CHAR | BYTE])

Creates a variable of type VARCHAR2 with length of up to *n* bytes or *n* characters. The maximum that *n* can be is 32k bytes (see note), and the minimum is 1 byte or 1 character. The maximum *n* for a VARCHAR2 variable with character semantics is determined by the number of bytes required to store each character for the chosen character set, with an upper limit of 32k bytes. The length semantics are determined by the length qualifiers CHAR or BYTE, and if not explicitly stated, the value of the NLS_LENGTH_SEMANTICS environment variable is applied to the bind variable. Explicitly

stating the length semantics at variable definition stage will always take precedence over the `NLS_LENGTH_SEMANTICS` setting.

> **Note:**
>
> If the client character `NLS_LANG` environment variable is not set and the database character set is multibyte, for example, AL32UTF32, then PL/SQL will truncate the data when the declared bind variable is smaller than the data returned from PL/SQL. For more information, see Limitation.

> **Note:**
>
> By default, the maximum `VARCHAR2` length is 4000 bytes. Attempting to use a maximum length greater than 4000 bytes raises *ORA-01460 : unimplemented or unreasonable conversion requested*
>
> To enable 32k maximum length, you must add the `MAX_STRING_SIZE=extended` parameter to your `init.ora` file.

`NVARCHAR2 (`*n*`)`

Creates a variable of type `NVARCHAR2` with length of up to *n* characters. The maximum that *n* can be is determined by the number of bytes required to store each character for the chosen national character set, with an upper limit of 32k bytes (see note). The only exception to this is when a SQL*Plus session is connected to a pre Oracle9*i* server, or the `SQLPLUSCOMPATIBILITY` system variable is set to a version less than `9.0.0`. In this case the length *n* can be in bytes or characters depending on the chosen national character set, with the upper limit of 32k bytes still retained.

> **Note:**
>
> By default, the maximum `NVARCHAR2` length is 4000 bytes. Attempting to use a maximum length greater than 4000 bytes raises *ORA-01460 : unimplemented or unreasonable conversion requested*
>
> To enable 32k maximum length, you must add the `MAX_STRING_SIZE=extended` parameter to your `init.ora` file.

`CLOB`

Creates a variable of type `CLOB`.

`NCLOB`

Creates a variable of type `NCLOB`.

`REFCURSOR`

Creates a variable of type `REF CURSOR`.

`BINARY_FLOAT`

Creates a variable of type `BINARY_FLOAT`. `BINARY_FLOAT` is a floating-point number that conforms substantially with the Institute for Electrical and Electronics Engineers (IEEE) Standard for Binary Floating-Point Arithmetic, IEEE Standard 754-1985.

`BINARY_DOUBLE`

Creates a variable of type `BINARY_DOUBLE`. `BINARY_DOUBLE` is a floating-point number that conforms substantially with the Institute for Electrical and Electronics Engineers (IEEE) Standard for Binary Floating-Point Arithmetic, IEEE Standard 754-1985.

`BOOLEAN`

Creates a variable of type `BOOLEAN`.

`VECTOR`

Creates a variable of type `VECTOR`. When you declare a bind variable of a `VECTOR` data type, you do not need to specify the dimension or vector format. It uses the default flexible dimension and vector format.

> **See Also:**
>
> Oracle AI Vector Search User's Guide

While inserting or updating a value in a `VECTOR` data type column using a `VECTOR` bind variable, if the vector text assigned to the bind variable does not match the `VECTOR` data type column defined in the database table, an error is displayed.

Usage

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. See PRINT for more information.

To automatically display the value of a bind variable created with `VARIABLE`, use the `SET AUTOPRINT` command. See SET AUTOP[RINT] {ON | OFF} for more information.

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 when you exit SQL*Plus.

To free resources used by `CLOB` and `NCLOB` bind variables, you may need to manually free temporary LOBs with:

```
EXECUTE DBMS_LOB.FREETEMPORARY(:cv)
```

All temporary LOBs are freed when you exit SQL*Plus.

SQL*Plus `SET` commands, such as `SET LONG` and `SET LONGCHUNKSIZE`, and `SET LOBOFFSET` can be used to control the size of the buffer while `PRINT`ing `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 Cursor Variables.

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 `PRINT`ing a `REFCURSOR`.

A `REFCURSOR` bind variable cannot be `PRINT`ed more than once without re-executing the PL/SQL `OPEN ... FOR` statement.

Examples

The following example illustrates creating a bind variable, changing its value, and displaying its current value.

To create a bind variable, you can use the `VARIABLE` command:

```
VARIABLE ret_val NUMBER
```

To change this bind variable in SQL*Plus, you must use a PL/SQL block:

```
BEGIN
 :ret_val:=4;
END;
/


PL/SQL procedure successfully completed.
```

To display the value of the bind variable in SQL*Plus, you can use the `PRINT` command:

```
PRINT ret_val
```

The following output is displayed:

```
   RET_VAL
----------
         4
```

The following example illustrates creating a bind variable, and then setting it to the value returned by a function:

```
VARIABLE id NUMBER
BEGIN
  :id := EMP_MANAGEMENT.HIRE
  ('BLAKE','MANAGER','KING',2990,'SALES');
END;
/
```

The value returned by the stored procedure is being placed in the bind variable, `:id`. It can be displayed using the `PRINT` command or can be used in the subsequent PL/SQL subprograms.

The following example illustrates displaying a bind variable automatically using the `SET AUTOPRINT` command :

```
SET AUTOPRINT ON
VARIABLE a REFCURSOR
BEGIN
  OPEN :a FOR SELECT LAST_NAME, CITY, DEPARTMENT_ID
  FROM EMP_DETAILS_VIEW
  WHERE SALARY > 12000
  ORDER BY DEPARTMENT_ID;
END;
/
```

There is no need to issue a `PRINT` command to display the variable.

```
PL/SQL procedure successfully completed.
LAST_NAME                    CITY
DEPARTMENT_ID
------------------------  ------------------------------
-------------
Hartstein                 Toronto
20
Russell                   Oxford
80
Partners                  Oxford
80
King                      Seattle
90
Kochhar                   Seattle
90
De Haan                   Seattle
90

6 rows selected.
```

The following example illustrates creating some variables:

```
VARIABLE id NUMBER
VARIABLE txt CHAR (20)
VARIABLE abc BOOLEAN
VARIABLE var1 VECTOR
VARIABLE myvar REFCURSOR
```

You can execute the `VARIABLE` command without arguments to list the defined variables:

```
VARIABLE
```

The following output is displayed:

```
variable    id
datatype    NUMBER
```

```
variable    txt
datatype    CHAR(20)

variable    abc
datatype    BOOLEAN

variable    var1
datatype    VECTOR

variable    myvar
datatype    REFCURSOR
```

The following example illustrates listing a single variable:

```
VARIABLE txt
```

The following output is displayed:

```
variable txt
datatype CHAR(20)
```

The following example illustrates assigning a value to a variable for input binding:

```
VARIABLE tmp_var VARCHAR2(10)=Smith
```

The following example illustrates an alternate method to achieve the same result as the previous example:

```
VARIABLE tmp_var VARCHAR2(10)
VARIABLE tmp_var=Smith
EXECUTE DBMS_OUTPUT.PUT_LINE(:tmp_var)
```

The following example illustrates producing a report listing individual salaries and computing the departmental salary cost for employees who earn more than $12,000 per month:

```
VARIABLE rc REFCURSOR
BEGIN
  OPEN :rc FOR SELECT DEPARTMENT_NAME, LAST_NAME, SALARY
  FROM EMP_DETAILS_VIEW
  WHERE SALARY > 12000
  ORDER BY DEPARTMENT_NAME, LAST_NAME;
END;
/


PL/SQL procedure successfully completed.
```

You can set the page size to display the output:

```
SET PAGESIZE 100 FEEDBACK OFF
TTITLE LEFT '*** Departmental Salary Bill ***' SKIP 2
COLUMN SALARY FORMAT $999,990.99 HEADING 'Salary'
COLUMN DEPARTMENT_NAME HEADING 'Department'
COLUMN LAST_NAME HEADING 'Employee'
COMPUTE SUM LABEL 'Subtotal:' OF SALARY ON DEPARTMENT_NAME
COMPUTE SUM LABEL 'Total:' OF SALARY ON REPORT
BREAK ON DEPARTMENT_NAME SKIP 1 ON REPORT SKIP 1
PRINT rc
```

ORACLE

The following output is displayed:

```
*** Departmental Salary Bill ***

DEPARTMENT_NAME                 Employee                   Salary
------------------------------- -------------------------- ------------
Executive                       De Haan                      $17,000.00
                                King                         $24,000.00
                                Kochhar                      $17,000.00
****************************                               ------------
Subtotal:                                                    $58,000.00

Marketing                       Hartstein                    $13,000.00
****************************                               ------------
Subtotal:                                                    $13,000.00

Sales                           Partners                     $13,500.00
                                Russell                      $14,000.00
****************************                               ------------
Subtotal:                                                    $27,500.00

                                                           ------------
Total:                                                       $98,500.00
```

The following example illustrates creating an input bind to insert CLOB data into a CLOB
column:

```
SQL> CREATE TABLE xyz (col1 CLOB);
Table created.

SQL> VARIABLE abc VARCHAR2(100)="This is a clob input"
SQL> INSERT INTO xyz VALUES (:abc);
1 row created.
```

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.
```

You can produce a report listing the data in the col_clob column:

```
VARIABLE T CLOB
BEGIN
  SELECT CLOB_COL INTO :T FROM CLOB_TAB;
END;
/
PL/SQL PROCEDURE SUCCESSFULLY COMPLETED
```

You can print 200 characters from the column clob_col using the following command:

```
SET LINESIZE 70
SET LONG 200
PRINT T
```

```
T
----------------------------------------------------------------------
Remember to run the Departmental Salary Bill report each month This report
contains confidential information.
```

You can set the printing position to the 21st character using the following command:

```
SET LOBOFFSET 21
PRINT T

T
----------------------------------------------------------------------
Departmental Salary Bill report each month This report contains
confidential information.
```

You can reference a bind variable of the `BOOLEAN` data type in a PL/SQL block:

```
VARIABLE abc BOOLEAN
BEGIN
  :abc := 'TRUE';
END;
/
```

You can display the referenced bind variable using a `SELECT` or a `PRINT` command:

```
SELECT :abc FROM DUAL;

:ABC
---------
TRUE

PRINT :abc

:ABC
---------
TRUE
```

You can insert data into `BOOLEAN` columns:

```
VARIABLE var1 BOOLEAN='TRUE'
VARIABLE var2 BOOLEAN='FALSE'
INSERT INTO my_table Values (:var1, :var2);
```

You can use a `SELECT` command to display the `BOOLEAN` column data:

```
SELECT * FROM my_table;
```

The following output is displayed:

```
COL1      COL2
------    ------
TRUE      FALSE
```

You can insert data into a `VECTOR` column:

```
VARIABLE var1 VECTOR='[2,3]'
INSERT INTO my_vector1 VALUES(:var1);
1 row created
```

You can use a `SELECT` command to display the `VECTOR` column data:

```
SELECT * FROM my_vector1;
```

The following output is displayed:

```
INT
--------------------------------------------------------------------------------
[2.0E+000,3.0E+000]
```

You can reference a bind variable of the `VECTOR` data type in a PL/SQL block:

```
SET SERVEROUTPUT ON;
VARIABLE plsql_flt32 VECTOR='[8.4,9.6]'
BEGIN
  INSERT INTO myvector_2 VALUES(:plsql_flt32);
END;
/
PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.
```

You can use a `SELECT` command to display the `VECTOR` column data:

```
SELECT * FROM my_vector2;
```

The following output is displayed:

```
FLT32
--------------------------------------------------------------------------------
[8.39999962E+000,9.60000038E+000]
```

13.57 WHENEVER OSERROR

Syntax

```
WHENEVER OSERROR {EXIT [SUCCESS | FAILURE | n | variable | :BindVariable]
[COMMIT | ROLLBACK] | CONTINUE [COMMIT | ROLLBACK | NONE]}
```

Performs the specified action (exits SQL*Plus by default) if an operating system error occurs (such as a file writing error).

Terms

```
[SUCCESS | FAILURE | n | variable | :BindVariable]
```

Directs SQL*Plus to perform the specified action 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.

```
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 for more information.

```
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

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.

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

If a failure occurs when reading from the output file, the commands in the following script cause SQL*Plus to exit and COMMIT any pending changes:

```
WHENEVER OSERROR EXIT
START no_such_file


OS Message: No such file or directory
Disconnected from Oracle......
```

13.58 WHENEVER SQLERROR

Syntax

```
WHENEVER SQLERROR {EXIT [SUCCESS | FAILURE | WARNING | n | variable  | :BindVariable]
[COMMIT | ROLLBACK] | CONTINUE [COMMIT | ROLLBACK | NONE]}
```

Performs the specified action (exits SQL*Plus by default) if a SQL command or PL/SQL block generates an error.

Terms

```
[SUCCESS | FAILURE | WARNING | n | variable | :BindVariable]
```

Directs SQL*Plus to perform the specified action 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.

```
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 for more information.

```
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

The WHENEVER SQLERROR command is triggered by SQL command or PL/SQL block errors, and not by SQL*Plus command errors.

Examples

The commands in the following script cause SQL*Plus to exit and return the SQL error code if the SQL UPDATE command fails:

```
WHENEVER SQLERROR EXIT SQL.SQLCODE
UPDATE EMP_DETAILS_VIEW SET SALARY = SALARY*1.1;
```

The following examples show that the WHENEVER SQLERROR command is not executed after errors with SQL*Plus commands, but it is executed if SQL commands or PL/SQL blocks cause errors:

```
WHENEVER SQLERROR EXIT SQL.SQLCODE
column LAST_name headIing "Employee Name"


Unknown COLUMN option "headiing"


SHOW non_existed_option
```

The following PL/SQL block error causes SQL*Plus to exit and return the SQL error code:

```
WHENEVER SQLERROR EXIT SQL.SQLCODE
begin
  SELECT COLUMN_DOES_NOT_EXIST FROM DUAL;
```

```
END;
/

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.....
```

13.59 XQUERY

Syntax

```
XQUERY xquery_statement
```

The SQL*Plus XQUERY command enables you to perform an XQuery 1.0 query on a specified database. XQUERY is supported on Oracle Database 10*g* (Release 2) and later versions. Attempting to use XQUERY on an earlier version of the Oracle Database gives the error:

```
SP2-614 Server version too low
```

Terms

```
xquery_statement
```

Specifies the XQuery statement you want to run. The statement is entered with standard XQuery syntax. The XQUERY statement is terminated with a forward slash, '/'.

Usage

Prefix your XQuery statement with the SQL*Plus command, XQUERY, and terminate the XQUERY command with a slash (/). XQUERY is a SQL*Plus keyword. If XQueries are executed in other tools, the keyword may not be needed.

XML output from the XQUERY command is displayed as native XML according to the active SET command options. SET LONG typically needs to be set. It may be useful to consider the following settings:

- Linesize for rows longer than the default 80 characters (SET LINESIZE).

- LOB, LONG and XML Type Size for rows longer than the default 80 characters (SET LONG).

- Output Page Setup to match output (SET PAGESIZE).

- Display Headings to repress the "Result Sequence" column heading (SET HEADING OFF).

The XQUERY command requires an active database connection. The command will not work with SQLPLUS /NOLOG.

Bind variables are not supported in the XQUERY command.

There are four SET commands specific to the XQUERY command. The SHOW XQUERY command gives the status of these settings. They are:

- SET XQUERY BASEURI {text}

- SET XQUERY ORDERING {UNORDERED | ORDERED | DEFAULT}

- SET XQUERY NODE {BYVALUE | BYREFERENCE | DEFAULT}

- SET XQUERY CONTEXT {text}

Examples

The XQuery statement in the following script queries the EMP_DETAILS_VIEW view of the HR schema:

```
set long 160
set linesize 160
xquery for $i in fn:collection("oradb:/SCOTT/EMP_DETAILS_VIEW") return $i
/

Result Sequence
--------------------------------------------------------------------
--------------------
<ROW><EMPNO>7369</EMPNO><ENAME>SMITH</ENAME><JOB>CLERK</JOB><MGR>7902</
MGR><HIREDATE>17-
DEC-80</HIREDATE><SAL>800</SAL><DEPTNO>20</DEPTNO></ROW>

14 item(s) selected.
```

Part IV

SQL*Plus Appendixes

Part IV contains the following SQL*Plus appendixes:

- SQL*Plus Limits
- SQL*Plus COPY Command
- Obsolete SQL*Plus Commands
- SQL*Plus Instant Client

ORACLE®

A
SQL*Plus Limits

The general SQL*Plus limits shown are valid for most operating systems.

Table A-1   SQL*Plus Limits

Item	Limit
filename length	system dependent
username length	128 bytes
substitution variable name length	128 bytes
substitution variable value length	240 characters
command-line length	5000 characters
LONG	2,000,000,000 bytes
LINESIZE	system dependent
LONGCHUNKSIZE value	system dependent
output line size	system dependent
SQL or PL/SQL command- 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	32,767 characters
maximum ARRAYSIZE	5000 rows
maximum number of nested scripts	20
maximum page number	99,999
maximum PL/SQL error message size	2K
maximum ACCEPT character string length	240 Bytes
maximum number of substitution variables	2048

B

SQL*Plus COPY Command

This appendix discusses the following topics:

- COPY Command Syntax
- Copying Data from One Database to Another
- About 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 SQL*Plus Quick Start .

The COPY command will be deprecated in future releases of SQL*Plus. After Oracle 9*i*, no new datatypes are supported by COPY.

B.1 COPY Command Syntax

```
COPY {FROM database | TO database | FROM database TO database}          {APPEND|
CREATE|INSERT|REPLACE} destination_table [(column, column, column, ...)]  USING query
```

where *database* has the following syntax:

```
username[/password]@connect_identifier
```

Copies data from a query to a table in the same or another database. COPY supports the following datatypes:

```
CHAR
DATE
LONG
NUMBER
VARCHAR2
```

> ⚠ **WARNING:**
>
> Including your password in plain text is a security risk. You can avoid this risk by omitting the password, and entering it only when the system prompts for it.

B.1.1 Terms

See the following list for a description of each term or clause:

```
FROM database
```

The 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 use a FROM clause to specify a source database other than

the default. The COPY command FROM clause does not support SYSDBA or SYSOPER privileged connections.

`TO database`

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 use a TO clause to specify a destination database other than the default. The COPY command TO clause does not support SYSDBA or SYSOPER privileged connections.

`database`

Specifies *username*[/*password*] *@connect_identifier* of the Oracle Database source or destination database you wish to COPY FROM or COPY TO. The COPY command does not support SYSDBA or SYSOPER privileged connections. You must include a username. SQL*Plus prompts you for the password associated with the username specified in the COPY FROM or COPY TO clauses. SQL*Plus suppresses the display of your password response.

You must include the *connect_identifier* clause to specify the source or destination database. The exact syntax depends on the Oracle Net configuration. For more information, refer to the Oracle Net manual or contact your DBA.

`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 *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.

`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.

B.1.2 Usage

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 LONG variable 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 COPYCOMMIT variable to a positive value n, SQL*Plus performs a commit after copying every n batches of records. The SQL*Plus SET ARRAYSIZE variable determines the size of a batch.

Some operating environments require that service names be placed in double quotes.

B.1.3 Examples

The following command copies the entire EMPLOYEES table to a table named WESTEMPLOYEES. Note that the tables are located in two different databases. If WESTEMPLOYEES already exists, SQL*Plus replaces the table and its contents. The columns in WESTEMPLOYEES have the same names as the columns in the source table, EMPLOYEES.

```
COPY FROM HR@HQ TO JOHN@WEST -
REPLACE WESTEMPLOYEES -
USING SELECT * FROM EMPLOYEES
```

The following command copies selected records from EMPLOYEES to the database to which SQL*Plus is connected. SQL*Plus creates SALESMEN through the copy. SQL*Plus copies only the columns EMPLOYEE_ID and LAST_NAME, and at the destination names them EMPLOYEE_ID and SA_MAN.

```
COPY FROM HR@ORACLE01 -
CREATE SALESMEN (EMPLOYEE_ID, SA_MAN) -
USING SELECT EMPLOYEE_ID, LAST_NAME FROM EMPLOYEES -
WHERE JOB_ID='SA_MAN';
```

B.2 Copying Data from One Database to Another

Use the SQL*Plus COPY command to copy CHAR, DATE, LONG, NUMBER or VARCHAR2 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 (most systems).
- Copy data from one remote database to another remote database (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.

B.2.1 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 HR@BOSTONDB -
TO TODD@CHICAGODB -
CREATE NEWDEPT (DEPARTMENT_ID, DEPARTMENT_NAME, CITY) -
USING SELECT * FROM EMP_DETAILS_VIEW
```

To specify a database in the FROM or TO clause, you must have a valid username and password for the local and remote databases and know the appropriate Oracle Net service names. COPY obeys Oracle Database 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. However, including both clauses increases the readability of your scripts.

The COPY command behaves differently based on whether the destination table already exists and on the action clause you enter (CREATE in the example). See About Controlling Treatment of the Destination Table for more information.

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 DEPARTMENT_ID is 30:

```
COPY FROM HR@BOSTONDB -
REPLACE EMPCOPY2 -
USING SELECT LAST_NAME, SALARY -
FROM EMP_DETAILS_VIEW -
WHERE DEPARTMENT_ID = 30
```

You may find it easier to enter and edit long COPY commands in scripts rather than trying to enter them directly at the command prompt.

B.2.2 About 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.

> **Note:**
>
> See your DBA for an appropriate username, password, and service name for a remote computer that contains a copy of EMPLOYEE_COPY.

```
COPY FROM HR@BOSTONDB -
CREATE EMPCOPY -
USING SELECT * FROM HR
```

```
Array fetch/bind size is 15. (arraysize is 15)
Will commit when done. (copycommit is 0)
Maximum long size is 80. (long is 80)
```

SQL*Plus then creates the table EMPLOYEE_COPY and copies the rows:

```
Table SALESMAN created.

   5 rows selected from HR@BOSTONDB.
   5 rows inserted into SALESMAN.
   5 rows committed into SALESMAN at DEFAULT HOST connection.
```

In this COPY command, the FROM clause directs COPY to connect you to the database with the specification BOSTONDB as HR.

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 line continuation hyphen (-), optionally preceded by a space, at the end of each line except the last.

Example B-1   Copying from a Remote Database to Your Local Database Using CREATE

To copy HR from a remote database into a table called EMPLOYEE_COPY on your own database, enter the following command:

B.2.3 About 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 SET variables, including how to change their settings, see the SET command.

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.

B.2.4 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 DEPARTMENT owned by the username ADAMS on the database associated with the Oracle Net connect identifier BOSTONDB, you would enter

```
COPY FROM HR@BOSTONDB -
CREATE EMPLOYEE_COPY2 -
USING SELECT * FROM ADAMS.DEPARTMENT
```

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.

B.3 About 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 the service name for your local database in either a FROM or a TO clause (omit the other clause):

```
COPY FROM HR@MYDATABASE -
INSERT EMPLOYEE_COPY2 -
USING SELECT * FROM EMPLOYEE_COPY
```

To copy between tables on a remote database, include the same username and service name in the FROM and TO clauses:

```
COPY FROM HR@BOSTONDB -
TO HR@BOSTONDB -
INSERT EMPLOYEE_COPY2 -
USING SELECT * FROM EMPLOYEE_COPY
```

C

Obsolete SQL*Plus Commands

This appendix covers earlier versions of some SQL*Plus commands. While these older commands still function in SQL*Plus, they are not supported. It is recommended that you use the alternative SQL*Plus commands listed in the following table.

C.1 SQL*Plus Obsolete Command Alternatives

Obsolete commands are available in current releases of SQL*Plus. In future releases, they may only be available by setting the SQLPLUSCOMPATIBILITY variable. You should modify scripts using obsolete commands to use the alternative commands.

Obsolete Command	Alternative Command	Description of Alternative Command
BTITLE (old form)	BTITLE	Places and formats a title at the bottom of each report page or lists the current BTITLE definition.
COLUMN DEFAULT	COLUMN CLEAR	Resets column display attributes to default values.
DOCUMENT	REMARK	Places a comment which SQL*Plus does not interpret as a command.
NEWPAGE	SET NEWP[AGE] {1 | n | NONE}	Sets the number of blank lines to be printed from the top of each page to the top title.
SET BUFFER	EDIT	Enables the editing of the SQL*Plus command buffer, or the contents of a saved file. Use the SQL*Plus SAVE, GET, @ and START commands to create and use external files.
SET COMPATIBILITY	none	Obsolete
SET CLOSECURSOR	none	Obsolete
SET DOCUMENT	none	Obsolete
SET MAXDATA	none	Obsolete
SET SCAN	SET DEF[INE] {& | c | ON | OFF}	Sets the character used to prefix substitution variables.
SET SPACE	SET COLSEP { | text}	Sets the text to be printed between SELECTed columns.
SET TRUNCATE	SET WRA[P] {ON | OFF}	Controls whether SQL*Plus truncates a SELECTed row if it is too long for the current line width.

Obsolete Command	Alternative Command	Description of Alternative Command
SHOW LABEL	none	Obsolete
TTITLE (old form)	TTITLE	Places and formats a title at the top of each report page or lists the current TTITLE definition.

C.2 BTITLE (old form)

Syntax

```
BTI[TLE] text
```

Displays a title at the bottom of each report page.

The old form of BTITLE offers formatting features more limited than those of the new form, but provides compatibility with UFI (a predecessor of SQL*Plus). The old form defines the bottom title as an empty line followed by a line with centered text. See TTI[TLE] text (obsolete old form) for more details.

C.3 COLUMN DEFAULT

Syntax

```
COL[UMN] {column|expr} DEF[AULT] (obsolete)
```

Resets the display attributes for a given column to default values.

Has the same effect as COLUMN CLEAR.

C.4 DOCUMENT

Syntax

```
DOC[UMENT]
```

Begins a block of documentation in a script.

For information on the current method of inserting comments in a script, see the section About Placing Comments in Scripts and the REMARK command.

After you type DOCUMENT and enter [Return], SQL*Plus displays the prompt DOC> in place of SQL> until you end the documentation. The "pound" character (#) on a line by itself ends the documentation.

If you have set DOCUMENT to OFF, SQL*Plus suppresses the display of the block of documentation created by the DOCUMENT command. For more information, see SET DOC[UMENT] {ON|OFF} (obsolete).

C.5 NEWPAGE

Syntax

```
NEWPAGE [1|n]
```

Advances spooled output n lines beyond the beginning of the next page.

See SET NEWP[AGE] {1 | n | NONE} for information on the current method for advancing spooled output.

C.6 SET BUFFER

Syntax

```
SET BUF[FER] {buffer|SQL}
```

Makes the specified *buffer* the current buffer.

Initially, the SQL buffer is the current buffer. SQL*Plus does not require the use of multiple buffers; the SQL buffer alone should meet your needs.

If the buffer name you enter does not exist, SET BUFFER defines (creates and names) the buffer. SQL*Plus deletes the buffer and its contents when you exit SQL*Plus.

Running a query automatically makes the SQL buffer the current buffer. To copy text from one buffer to another, use the GET and SAVE commands. To clear text from the current buffer, use CLEAR BUFFER. To clear text from the SQL buffer while using a different buffer, use CLEAR SQL.

C.7 SET COMPATIBILITY

Syntax

```
SET COM[PATIBILITY]{V7 | V8 | NATIVE}
```

Specifies the version of the SQL language parsing syntax to use.

Set COMPATIBILITY to V7 for Oracle7, or to V8 for Oracle8 or later. COMPATIBILITY always defaults to NATIVE. Set COMPATIBILITY for the version of Oracle Database SQL syntax you want to use on the connected database, otherwise.

The default compatibility setting, NATIVE, is the most relevant setting for modern Oracle databases.

For information about SQL*Plus version compatibility settings, see SET SQLPLUSCOMPAT[IBILITY] {x.y[.z]}.

Example

To run a script, SALARY.SQL, created with Oracle7 SQL syntax, enter

```
SET COMPATIBILITY V7
START SALARY
```

After running the file, reset compatibility to NATIVE to run scripts created for Oracle Database 10*g*:

```
SET COMPATIBILITY NATIVE
```

Alternatively, you can add the command SET COMPATIBILITY V7 to the beginning of the script, and reset COMPATIBILITY to NATIVE at the end of the file.

C.8 SET CLOSECURSOR

Syntax

```
SET CLOSECUR[SOR] {ON|OFF}
```

Sets the cursor usage behavior.

On or OFF sets whether or not the cursor will close and reopen after each SQL statement. This feature may be useful in some circumstances to release resources in the database server.

C.9 SET DOCUMENT

Syntax

```
SET DOC[UMENT] {ON|OFF}
```

Displays or suppresses blocks of documentation created by the DOCUMENT command.

SET DOCUMENT ON causes blocks of documentation to be echoed to the screen. Set DOCUMENT OFF suppresses the display of blocks of documentation.

See DOC[UMENT] (obsolete) for information on the DOCUMENT command.

C.10 SET MAXDATA

Syntax

```
SET MAXD[ATA] n
```

Sets the maximum total row width that SQL*Plus can process.

In SQL*Plus, the maximum row width is now unlimited. Any values you set using SET MAXDATA are ignored by SQL*Plus.

C.11 SET SCAN

Syntax

```
SET SCAN {ON|OFF}
```

Controls scanning for the presence of substitution variables and parameters. OFF suppresses processing of substitution variables and parameters; ON enables normal processing.

ON functions in the same manner as SET DEFINE ON.

C.12 SET SPACE

Syntax

```
SET SPACE {1|n}
```

Sets the number of spaces between columns in output. The maximum value of n is 10.

The SET SPACE 0 and SET COLSEP " commands have the same effect. This command is obsoleted by SET COLSEP, but you can still use it for backward compatibility. You may prefer to use COLSEP because the SHOW command recognizes COLSEP and does not recognize SPACE.

C.13 SET TRUNCATE

Syntax

```
SET TRU[NCATE] {ON|OFF}
```

Controls whether SQL*Plus truncates or wraps a data item that is too long for the current line width.

ON functions in the same manner as SET WRAP OFF, and vice versa. You may prefer to use WRAP because the SHOW command recognizes WRAP and does not recognize TRUNCATE.

C.14 TTITLE (old form)

Syntax

```
TTI[TLE] text
```

Displays a title at the top of each report page.

The old form of TTITLE offers formatting features more limited than those of the new form, but provides compatibility with UFI (a predecessor of SQL*Plus). The old form defines the top

title as a line with the date left-aligned and the page number right-aligned, followed by a line with centered text and then a blank line.

The text you enter defines the title TTITLE displays.

SQL*Plus centers text based on the size of a line as determined by SET LINESIZE. A separator character (|) begins a new line; two line separator characters in a row (||) insert a blank line. You can change the line separator character with SET HEADSEP.

You can control the formatting of page numbers in the old forms of TTITLE and BTITLE by defining a variable named "_page". The default value of _page is the formatting string "page &P4". To alter the format, you can DEFINE _page with a new formatting string as follows:

```
SET ESCAPE / SQL> DEFINE _page = 'Page /&P2'
```

This formatting string will print the word "page" with an initial capital letter and format the page number to a width of two. You can substitute any text for "page" and any number for the width. You must set escape so that SQL*Plus does not interpret the ampersand (&) as a substitution variable. See SET ERRORL[OGGING] {ON | OFF} [TABLE [schema.]tablename] [TRUNCATE] [IDENTIFIER identifier] for more information on setting the escape character.

SQL*Plus interprets TTITLE in the old form if a valid new-form clause does not immediately follow the command name.

If you want to use CENTER with TTITLE and put more than one word on a line, you should use the new form of TTITLE. For more information see the TTITLE command.

Example

To use the old form of TTITLE to set a top title with a left-aligned date and right-aligned page number on one line followed by SALES DEPARTMENT on the next line and PERSONNEL REPORT on a third line, enter

```
TTITLE 'SALES DEPARTMENT|PERSONNEL REPORT'
```

D

SQL*Plus Instant Client

SQL*Plus Instant Client is a standalone product with all the functionality of SQL*Plus command-line. It connects to existing remote Oracle databases, but does not include its own database. It is easy to install and uses significantly less disk space than the full Oracle Database Client installation required to use SQL*Plus command-line.

SQL*Plus Instant Client is available on platforms that support the OCI Instant Client. See About OCI Instant Client for more information on the OCI Instant Client.

To install SQL*Plus Instant Client, you need two packages:

- SQL*Plus Instant Client package.
- Either the Basic OCI Instant Client package, or the lightweight OCI Instant Client package.

D.1 About Choosing the SQL*Plus Instant Client to Install

SQL*Plus Instant Client can be installed in two ways:

- Download the packages from the Oracle Technology Network (OTN).
- Copy the same files that are in the packages from an Oracle Database 21*c* Client Administrator installation.

Both the SQL*Plus and OCI packages must be from the same Oracle Database version.

D.1.1 Basic Instant Client

SQL*Plus Instant Client using the Basic OCI package works with any NLS_LANG setting supported by the Oracle Database. It supports all character sets and language settings available in the Oracle Database.

D.1.2 Lightweight Instant Client

SQL*Plus Instant Client using the lightweight OCI package displays error messages in English only and supports only specific character sets. It is significantly smaller than SQL*Plus Instant Client using the Basic OCI package.

Valid values for NLS_LANG parameters with the lightweight Instant Client are:

- *language* can be any valid language supported by the Oracle Database, however, error messages are only reported in English.
- *territory* can be any valid territory supported by the Oracle Database.
- *charset* is one of the following character sets:
  - US7ASCII
  - WE8DEC
  - WE8MSWIN1252

- – WE8ISO8859P1
- – UTF8
- – AL16UTF16
- – AL32UTF8

For example:

```
NLS_LANG=AMERICAN_AMERICA.UTF8
```

See Setting Up a Globalization Support Environment, and NLS_LANG Environment Variable for more information about NLS settings.

D.1.2.1 Lightweight SQL*Plus Instant Client Error with Unsupported Character Set

Attempting to start SQL*Plus Instant Client with an unsupported character set will fail with the following error:

```
Error 5 initializing SQL*Plus
NLS initialization error
```

D.2 List of Files Required for SQL*Plus Instant Client

This section lists the required files for installation of SQL*Plus Instant Client for both the basic package and the lightweight package.

The files from only one of the OCI packages (either basic or lightweight) are required. Other files that get installed, which are not listed in this section, can be ignored or can be removed to save disk space.

Table D-1   Instant Client Files in the SQL*Plus Package

Linux and UNIX	Windows	Description
sqlplus	sqlplus.exe	SQL*Plus executable
libsqlplus.so	not applicable	SQL*Plus library
libsqlplusic.so	orasqlplusic21.dll	SQL*Plus data shared library

Table D-2   Instant Client Files in the Basic OCI Package

Linux and UNIX	Windows	Description
libclntsh.so.21.1	oci.dll	Client code library
libclntshcore.so	not applicable	OCI Instant Client data shared Library
libociei.so	oraociei21.dll	OCI Instant Client data shared library
libnnz21.so	orannzsbb21.dll	Security library
libons.so	oraons.dll	ONS library

Table D-3   Instant Client Files in the Lightweight OCI Package

Linux and UNIX	Windows	Description
libclntsh.so.21.1	oci.dll	Client code library
libociicus.so	oraociicus21.dll	OCI Instant Client data shared library (English only)
libnnz21.so	orannzsbb21.dll	Security library

D.3 Installing SQL*Plus Instant Client by Downloading Installation Files from OTN

This section describes how to install SQL*Plus Instant Client from OTN RPM packages for Linux and from OTN downloadable zip files for UNIX and Windows.

The SQL*Plus Instant Client package should never be installed on an `ORACLE_HOME`.

- From Linux RPM Packages
- From UNIX or Windows Zip Files

From Linux RPM Packages

1. Download the RPM packages containing the SQL*Plus Instant Client package and the OCI package from the OTN Instant Client page. Both packages must be of the same version.

   > **See Also:**
   >
   > Oracle Instant Client Page

2. Use either the `rpm -i` command for the initial installation of the RPM packages, or the `rpm -u` command to upgrade to a newer version of the packages.

3. Configure SQL*Plus Instant Client.

   > **See Also:**
   >
   > Configuring SQL*Plus Instant Client

From UNIX or Windows Zip Files

1. Download the zip files containing the SQL*Plus Instant Client package and the OCI package from the OTN Instant Client page. Both packages must be of the same version.

> **See Also:**
>
> Oracle Instant Client Page

2. Create a new directory, for example, `/home/instantclient20_2` on UNIX or `c:\instantclient20_2` on Windows.

3. Unzip the two packages into the new directory.

4. Configure SQL*Plus Instant Client.

> **See Also:**
>
> Configuring SQL*Plus Instant Client

D.4 Installing SQL*Plus Instant Client from the 21c Client Release Media

This section describes how to install SQL*Plus Instant Client on Unix, Linux, and Windows.

1. Run the installer on the Oracle Database 21c Client Release media and choose the Administrator option.

2. Create a new directory, for example, `/home/instantclient_21_1` on UNIX and Linux, or `c:\instantclient_21_1` on Windows.

3. Copy the SQL*Plus Instant Client and the OCI Instant Client files to the new directory. All files must be copied from the same `ORACLE_HOME`.

> **See Also:**
>
> List of Files Required for SQL*Plus Instant Client

4. Configure SQL*Plus Instant Client.

> **See Also:**
>
> Configuring SQL*Plus Instant Client

D.5 Configuring SQL*Plus Instant Client

You must use the SQL*Plus Instant Client executable only with the matching version of the OCI Instant Client.

> **Note:**
>
> You do not need to set the `ORACLE_HOME` or `ORACLE_SID` environment variables.

- Linux from RPMs
- Linux from Client Media or Zip File and UNIX
- Windows

Linux from RPMs

> **Note:**
>
> You do not need to set the `ORACLE_HOME` or `ORACLE_SID` environment variables.

You must download the RPMs from OTN into Oracle specific sub-directories in the `/usr` file system. The sub-directory structure enables multiple versions of Instant Client to be available.

> **Note:**
>
> You do not need to set the `ORACLE_HOME` or `ORACLE_SID` environment variables.

1. Add the name of the directory containing the Instant Client libraries to `LD_LIBRARY_PATH`. Remove any other Oracle directories.

   > **Note:**
   >
   > You do not need to set the `ORACLE_HOME` or `ORACLE_SID` environment variables.

   For example, use the following command to set `LD_LIBRARY_PATH` on Solaris in the Bourne or Korn shells:

   ```
   LD_LIBRARY_PATH=/usr/lib/oracle/21.1/client/lib:${LD_LIBRARY_PATH}
   export LD_LIBRARY_PATH
   ```

2. Make sure that the *sqlplus* executable installed from the RPM is the first entry in your `PATH`. To test, enter the *which sqlplus* command, which should return */usr/bin/sqlplus*. If

you do not see this output, then remove any other Oracle directories from `PATH`, or put */usr/bin* before any other SQL*Plus executables present in `PATH`, or use an absolute or relative path to start SQL*Plus.

For example, use the following command to set `PATH` in the bash shell:

```
PATH=/usr/bin:${PATH}
export PATH
```

If you install multiple versions of SQL*Plus, then you may need to change the symbolic link */usr/bin/sqlplus* to the version of SQL*Plus matching the libraries in `LD_LIBRARY_PATH`. For release 21.1, */usr/bin/sqlplus* is a symbolic link to the SQL*Plus binary at */usr/lib/oracle/21.1/client/bin/sqlplus*.

3. Set Oracle globalization variables required for your locale. A default locale is assumed if no variables are set. See Locale Data for more information.
   For example:

```
NLS_LANG=AMERICAN_AMERICA.UTF8
export NLS_LANG
```

Linux from Client Media or Zip File and UNIX

1. Add the name of the directory containing the Instant Client files to the appropriate shared library path `LD_LIBRARY_PATH`, `LIBPATH` or `SHLIB_PATH`. Remove any other Oracle directories.
   For example, use the following command on Solaris in the Bourne or Korn shells:

```
LD_LIBRARY_PATH=/home/instantclient_21_1:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH
```

2. Add the directory containing the Instant Client files to the `PATH` environment variable. If it is not set, then an absolute or relative path must be used to start SQL*Plus. Remove any other Oracle directories from `PATH`. For example:

```
PATH=/home/instantclient_21_1:${PATH}
export PATH
```

3. Set Oracle globalization variables required for your locale. A default locale will be assumed if no variables are set. See Locale Data for more information. For example:

```
NLS_LANG=AMERICAN_AMERICA.UTF8
export NLS_LANG
```

Windows

You can configure the environment for a session using the `SET` commands in a Windows command prompt or globally by setting the **Environment Variables** in **System Properties**. For example, to set environment variables in Windows 2000 using **System Properties**, open **System** from the **Control Panel**, click the **Advanced** tab and then click **Environment Variables**.

Now, perform the following steps:

1. Add the directory containing the Instant Client files to the `PATH` system environment variable. Remove any other Oracle directories from `PATH`. For example, add *c:\instantclient20_2* to the beginning of `PATH`.

2. Set Oracle globalization variables required for your locale. A default locale will be assumed if no variables are set. See Locale Data for more information. For

example, to set `NLS_LANG` for a Japanese environment, create a user environment variable `NLS_LANG` set to `JAPANESE_JAPAN.JA16EUC`.
If you have installed the lightweight Instant Client, see Lightweight Instant Client for information about supported `NLS_LANG` settings.

D.6 About Connecting to a Database with SQL*Plus Instant Client

SQL*Plus Instant Client is always remote from any database server. To connect to a database, you must specify the database using an Oracle Net connection identifier.

An example using an Easy Connection identifier for connecting to the HR schema in the MYDB database, running on *mymachine*, is:

```
sqlplus hr/your_password@\"//mymachine.mydomain:port/MYDB\"
```

Alternatively, you can use a Net Service Name:

```
sqlplus hr/your_password@MYDB
```

Net Service Names can be stored in a number of places, including LDAP. The use of LDAP is recommended to take advantage of the new features of Oracle Database 21c. See Local Naming Parameters in the tnsnames.ora File for more information.

If you want to use Net Service Names configured in a local Oracle Net `tnsnames.ora` file, then set the environment variable `TNS_ADMIN` to the directory containing the `tnsnames.ora` file. For example, on UNIX, if your `tnsnames.ora` file is in `/home/user1` and it defines the Net Service Name `MYDB2`, then use the following commands:

```
TNS_ADMIN=/home/user1
export TNS_ADMIN
sqlplus hr@MYDB2
```

If `TNS_ADMIN` is not set, then an operating system dependent set of directories is examined to find `tnsnames.ora`. This search path includes looking in the directory specified by the `ORACLE_HOME` environment variable for `network/admin/tnsnames.ora`. This is the only reason to set the `ORACLE_HOME` environment variable for SQL*Plus Instant Client. If `ORACLE_HOME` is set when running Instant Client applications, then it must be set to a directory that exists.

This example assumes the `ORACLE_HOME` environment variable is set, and the `$ORACLE_HOME/network/admin/tnsnames.ora` or `ORACLE_HOME\network\admin\tnsnames.ora` file defines the Net Service Name `MYDB3`:

```
sqlplus hr@MYDB3
```

You can set the `TWO_TASK` (on UNIX) or `LOCAL` (on Windows) environment variable to a connection identifier. This removes the need to explicitly enter the connection identifier whenever a connection is made in SQL*Plus or SQL*Plus Instant Client. This UNIX example connects to the database known as MYDB4:

```
TNS_ADMIN=/home/user1
export TNS_ADMIN
TWO_TASK=MYDB4
```

```
export TWO_TASK
sqlplus hr
```

On Windows, you can set the `TNS_ADMIN` and `LOCAL` variables in the System Properties.

> **✎ Note:**
>
> Configuring SQL*Plus Instant Client

D.7 AS SYSDBA or AS SYSOPER Connections with SQL*Plus Instant Client

To connect `AS SYSDBA` or `AS SYSOPER` to perform DBA tasks, you must set up an Oracle password file on the database server using the *orapwd* utility.

Once you configure the utility, your SQL*Plus Instant Client connection string looks like the following:

```
sqlplus sys@MYDB AS SYSDBA
```

See Using Password File Authentication for information on Oracle password files.

D.8 About Uninstalling Instant Client

The SQL*Plus Instant Client package can be removed separately from the OCI Instant Client. After uninstalling the SQL*Plus Instant Client package, the remaining OCI Instant Client libraries enable custom written OCI programs or third party database utilities to connect to a database.

D.8.1 Uninstalling SQL*Plus Instant Client

1. For installations on Linux from RPM packages, use *rpm -e* only on the SQL*Plus Instant Client package

   or

   For installations on UNIX and Windows, and installations on Linux from the Client Release media, manually remove the following SQL*Plus specific files:

   Table D-4   Instant Client Files in the SQL*Plus Package

UNIX	Windows	Description
sqlplus	sqlplus.exe	SQL*Plus executable
libsqlplus.so	not applicable	SQL*Plus library
libsqlplusic.so	orasqlplusic20.dll	SQL*Plus data shared library

2. Reset environment variables and remove *tnsnames.ora* if necessary.

D.8.2 Uninstalling the Complete Instant Client

1. For installations on Linux from RPM packages, use *rpm -qa* to find the SQL*Plus Instant Client and Basic OCI package names and run *rpm -e* to remove them

   or

   For installations on UNIX and Windows, and installations on Linux from the Client Release media, manually delete the directory containing the SQL*Plus executable and Oracle libraries.

   See Installing SQL*Plus Instant Client from the 21c Client Release Media for a list of the files you copied to the directory.

2. Reset environment variables such as PATH, SQLPATH, LD_LIBRARY_PATH and TNS_ADMIN.

3. Remove *tnsnames.ora* if necessary.

Index

Symbols

Numerics

A

file names *(continued)*
    in @@ (double at sign) command, *13-6*
    in EDIT command, *13-56*
    in GET command, *13-60*
    in SAVE command, *13-84*
    in SPOOL command, *7-27*, *13-148*
    in SQLPLUS command, *3-14*
files
    flat, *7-27*
FLAGGER variable, *13-87*, *13-110*
flat file, *7-27*
FLUSH variable, *9-9*, *13-87*, *13-110*
FOLD_AFTER clause, *13-25*
FOLD_BEFORE clause, *13-25*
font
    changing face and size in command-line, *1-1*
footers
    aligning elements, *13-82*
    displaying at bottom of page, *13-80*
    displaying system-maintained values, *13-82*
    formatting elements, *13-82*
    indenting, *13-82*
    listing current definition, *13-80*
    setting at the end of reports, *7-18*
    suppressing definition, *13-82*
FORCE clause, *13-152*
FORMAT clause, *13-8*, *13-25*
    in ATTRIBUTE command, *13-15*
    in COLUMN command, *7-3*, *7-5*
    in REPHEADER and REPFOOTER
        commands, *13-82*
    in TTITLE and BTITLE commands, *7-22*, *13-158*
format models, number, *7-4*, *13-27*
formfeed, to begin a new page, *7-25*, *13-122*
FROM clause, *13-74*, *B-4*

G

GET command, *13-59*
    LIST clause, *13-60*
    NOLIST clause, *13-60*
    retrieving scripts, *13-60*
globalization support, Oracle10g, *12-3*
glogin, *2-4*
    profile, *13-132*
    site profile, *2-4*, *2-5*, *3-13*, *13-131*
        See also login.sql

H

HEAD clause, *3-9*
HEAD option, *3-9*

headers
    aligning elements, *7-20*
    displaying at top of page, *13-81*
    displaying system-maintained values, *13-82*
    setting at the start of reports, *7-18*
    suppressing, *7-20*
HEADING clause, *7-1*, *13-28*
HEADING variable, *13-111*
headings
    aligning elements, *13-82*
    column headings, *13-111*
    formatting elements, *13-82*
    indenting, *13-82*
    listing current definition, *13-81*
    suppressing definition, *13-82*
HEADSEP variable, *13-87*, *13-111*
    use in COLUMN command, *7-2*
help
    installing command-line, *2-7*
    online, *3-6*, *13-60*
HELP command, ? command, *13-60*
HIDE clause, *13-9*
HISTORY clause, *13-140*
HISTORY command, *13-62*
    CLEAR clause, *13-63*
    EDIT clause, *13-62*
    DEL[ETE] clause, *13-62*
    LIST clause, *13-63*
    RUN clause, *13-62*
HISTORY variable, *13-87*
HOST command, *4-9*, *13-64*
HTML, *8-2*
    clause, *3-9*
    entities, *8-5*
    option, *3-9*
    spooling to file, *3-11*
    tag, *8-1*
hyphen
    continuing a long SQL*Plus command, *4-8*, *13-1*

I

IMMEDIATE mode, *13-147*
infinity sign (~), *13-27*
INIT.ORA file
    parameter file, *13-152*
initialization parameters
    displaying, *13-141*, *13-142*
input
    accepting values from the user, *6-23*, *13-8*
    accepting [Return], *6-26*
INPUT command, *5-3*, *5-6*, *13-65*
    entering several lines, *13-65*
INSERT clause, *B-2*, *B-5*

S