PART I

Preparing for OCP DBA Exam I: Introduction to SQL
Part I

Oracle9i Introduction to SQL Exam Objectives

The first exam in the OCP DBA and Developer tracks covers fundamental usage of the SQL programming language for interacting with the Oracle9i database. The following list identifies all OCP objectives for this exam as of this printing. This information comes from the OCP Candidate Guide published for this exam by Oracle Corporation. This candidate guide is online at www.oracle.com/education/certification:

1. Writing Basic SQL Select Statements
   1.1 List capabilities of SQL select statements
   1.2 Execute a basic select statement
   1.3 Differentiate between SQL and SQL*Plus commands

2. Restricting and Sorting Data
   2.1 Limit the rows retrieved by a query
   2.2 Sort the rows retrieved by a query

3. Single-Row Functions
   3.1 Describe various types of functions available in SQL
   3.2 Use character, number, and date functions in select statements
   3.3 Use conversion functions

4. Displaying Data from Multiple Tables
   4.1 Write select statements to access data from more than one table using equality and nonequality joins
   4.2 View data that generally does not meet a join condition by using outer joins
   4.3 Join a table to itself using a self-join

5. Aggregating Data Using Group Functions
   5.1 Identify the available group functions
   5.2 Use group functions
   5.3 Group data using the group by clause
   5.4 Include or exclude grouped rows using the having clause

6. Subqueries
   6.1 Describe the types of problems that subqueries can solve
   6.2 Define subqueries
6.3 List the types of subqueries
6.4 Write single-row and multiple-row subqueries

7. Producing Readable Output with SQL*Plus
   7.1 Produce queries that require a substitution variable
   7.2 Produce more readable output
   7.3 Create and execute script files

8. Manipulating Data
   8.1 Describe each DML statement
   8.2 Insert rows in a table
   8.3 Update rows in a table
   8.4 Delete rows in a table
   8.5 Merge rows in a table
   8.6 Control transactions

9. Creating and Managing Tables
   9.1 Describe the main database objects
   9.2 Create tables
   9.3 Describe the datatypes that can be used when specifying column definitions
   9.4 Alter table definitions
   9.5 Drop, rename, and truncate tables

10. Including Constraints
    10.1 Describe constraints
    10.2 Create and maintain constraints

11. Creating Views
    11.1 Describe a view
    11.2 Create, alter, and drop a view
    11.3 Retrieve data through a view
    11.4 Insert, update, and delete data through a view
    11.5 Create and use an inline view
    11.6 Perform top-N analysis

12. Creating Other Database Objects
    12.1 Create, maintain, and use sequences
    12.2 Create and maintain indexes
    12.3 Create private and public synonyms
13. Controlling User Access
   13.1 Create users
   13.2 Create roles to ease setup and maintenance of the security model
   13.3 Use the grant and revoke statements
CHAPTER 1

Overview of Oracle Databases
In this chapter, you will learn about and demonstrate knowledge in the following areas:

- Overview of Oracle databases
- Selecting data from Oracle

The first exam in the OCP series covers your understanding of basic areas of database usage and design. Every Oracle user, developer, and DBA should have complete mastery in these areas before moving into other test areas. This unit assumes little or no prior knowledge of Oracle on your part in order to help you go from never having used Oracle to having enough expertise in the Oracle server product to maintain and enhance existing applications and develop small new ones. This chapter will introduce Oracle and cover the basic aspects of data retrieval from the Oracle database.

This chapter covers material comprising approximately 8 percent of the test content of OCP Exam 1.

**Try Following along on Your Own Database!** As we move through the chapter, you will see examples of SQL statements issued on an Oracle database. For the most part, you can follow along with most of these examples on your own working database if you want. If the following instructions below look like a foreign language to you, show this page to your Oracle DBA and ask for his or her help:

1. On the command line of your machine hosting Oracle, change the directory to `$ORACLE_HOME/rdbms/admin`.
2. Log into Oracle as a privileged user, such as SYSTEM, who is allowed to create other users.
3. Issue the command `@utlsampl.sql`. This command runs the `utlsampl.sql` script, which creates objects owned by the user SCOTT/TIGER that we will use in the examples throughout the rest of the book.
4. If you’re more experienced with Oracle and want to specify your own username and password instead of using SCOTT/TIGER, you can run the `demobld.sql` script found in `$ORACLE_HOME/sqlplus/demo` while logged into Oracle as a user other than SCOTT.

**TIP**

Some of the more trivial examples in the chapter may use tables not created by `utlsampl.sql`. These examples are noted in the text. No script is available for creating those examples. If you want to
Overview of Oracle

This section covers the following topics as an overview of the Oracle database:

- Theoretical and physical aspects of relational databases
- Oracle's RDBMS and ORDBMS implementations
- Usage and benefits of PL/SQL

Welcome to the world of Oracle databases. Although this section is not associated with an official exam objective, it covers a great deal of the introductory material you may find helpful in order to get started with Oracle in preparation for using query operations to obtain data from the database. Many readers who have never used Oracle before find this material helpful in order to get the big picture of Oracle software before digging into the nitty-gritty. If you’re one of those readers, then read on! Even if you’re already a whiz at using Oracle SQL, you still might want to skim the material in this discussion before moving on, especially if you’ve never had an overview of Oracle software before. We’ll first talk about several basic aspects regarding theoretical and physical aspects of relational databases, as well as Oracle’s RDBMS and ORDBMS implementations. The use and benefits of PL/SQL—Oracle’s own language for developing database applications that are stored and executed directly inside the Oracle database—will be explained as well.

Theoretical and Physical Aspects of Relational Databases

Oracle finds its roots in relational database theory, as conceived by E. F. Codd in the 1950s, and extends those theories into an infinite variety of directions, such as data warehousing, online transaction processing, and Web-enabled applications. Undoubtedly, the popularity of this software is part of the reason you are reading this book. This book has the answers to your questions about what an Oracle database is, how it works, and what you can do with it, all of which you’ll need to know in order to pass the Introduction to SQL exam.

Software-development companies have taken many different approaches to information management. In years gone by, the more popular software packages for data storage and retrieval focused on flat-file systems as the storage means of choice while simultaneously requiring you to define how information is stored and retrieved, using a programming language such as COBOL. Some early breeds of flat-file systems included hierarchical storage systems, where data records were stored...
in a hierarchy similar to the hierarchical directory structure you might see on your PC's hard drive in Windows Explorer. These applications ran on mainframes, and brand names of these older data-management packages included IMS from IBM and IDMS from Computer Associates. The language most often used to develop mechanisms to add or manage data in those systems was COBOL.

Those older flat-file systems were great for certain tasks, such as defining parent/child relationships. A parent/child relationship might include the relationship of salespeople within a food service distribution company to the company's customers. Another parent/child relationship might be the tracking number for an invoice as it relates to product line items on the customer's order from that food service distribution company. However, one drawback to flat-file systems stems from the fact that a parent/child relationship cannot model every possible type of data relationship. Within the food service company example, a customer's order may list many different products. Each of those products themselves will probably appear on many different orders. In this case of a “many products to many orders” relationship, which way should the hierarchy be designed? What should be the parent and what should be the child? The usual solution was to create two separate hierarchies—one with product as parent, the other with order as parent. Unfortunately, this often meant maintaining much of the same information in two (or more) places, creating redundant data. Keeping data content consistent across multiple places where it is kept makes storage and retrieval complex. Another shortcoming of hierarchical databases using flat-file systems is that they are not easily adaptable to changing business needs. If the food service distributor creates a new sales system that calls for joint ownership of customer accounts by multiple salespeople, the hierarchical database needs to be redesigned.

Motivated by dissatisfaction with the cumbersome characteristics of hierarchical flat-file databases, E. F. Codd, a computer scientist working for IBM in the 1950s, developed an alternative: the relational model. Instead of storing data in hierarchies, Codd proposed storing related data items, such as control numbers and ordered products, in tables. If the tables were designed according to a few simple principles, they were both intuitive and extremely efficient in storing data, as Codd discovered. A single data item could be stored in only one place. Over time, many software makers recognized the significance of Codd's work and began developing products that adhered to Codd's model. Since the 1980s, virtually all database software products (including Oracle's) conform to the relational model.

Central to the success of the relational model is the use of a relational database management system, or RDBMS, for storing, retrieving, and manipulating data in a database. Earlier products required organizations to have many COBOL programmers on staff to code mechanisms for managing data-retrieval routines that interact directly with the files of the database. In contrast, the RDBMS handles these tasks automatically using a functional programming language called SQL (pronounced either sequel or as the letters spelled out). SQL stands for structured
query language, and it allows users to request the data they want according to strict comparison criteria. For example, if we wanted to look at an employee ID number and salary information for an employee named SMITH, the following code block shows a SQL statement that would help us do so:

```sql
SQL> SELECT EMPNO, ENAME, SAL FROM EMP
  2  WHERE ENAME = 'SMITH';
```

**TIP**
The preceding block was taken directly from SQL*Plus, a tool Oracle provides for interacting with the Oracle database. The “2,” which indicates that you are typing in the second line, is written automatically by SQL*Plus. As such, you do not actually need to type “2” yourself. For now, don’t worry about what this SQL statement actually does or what the results would be, just understand that it is an example of a SQL statement.

Behind the scenes, an RDBMS translates this statement into a series of operations that retrieve the actual data from a file somewhere on the machine hosting your database. This step is called parsing. After parsing is complete, the RDBMS executes the series of operations to complete the requested action. That series of operations may involve some or all of the following tasks, listed below in no particular order:

- Implicit datatype conversion
- Index lookups (if appropriate) for faster response time
- Disk reads or disk writes
- Filtering table data according to search criteria
- Sorting and formatting data returned

**TIP**
An index is a special database object that can be used to enhance performance of certain RDBMS operations. A datatype is literally a definition of the type of data being stored in the table’s column. You’ll learn more about both these topics in later chapters.
RDBMS vs. Flat-File System Quick Reference
Table 1-1 shows a quick comparison of flat-file systems to relational database management systems.

For Review
1. Understand the tasks an RDBMS completes behind the scenes when users request certain pieces of data.
2. Be sure you can describe the features, advantages, and disadvantages of flat-file systems and relational database management systems.

Exercises
1. You are exploring theoretical aspects of the Oracle RDBMS. Which of the following choices identifies an aspect of data management that the Oracle RDBMS does not handle on your behalf?
   A. Datatype conversion
   B. Disk reads
   C. Sorting and formatting return data
   D. Defining required information via SQL

2. You are evaluating the use of Oracle to replace legacy pre-relational systems in your organization. In comparison to the Oracle RDBMS, which of the following aspects of pre-relational database systems did those systems handle as well as their relational counterpart?
   A. Many-to-many data relationships
   B. Parent-child relationships
   C. Adaptability to changing business needs
   D. Data manipulation

3. What is the name of the scientist who first conceptualized the use of relational database management systems? ________________________

Answer Key
1. D 2. B 3. E. F. Codd
Oracle’s RDBMS and ORDBMS Implementations

Although every relational database offers an RDBMS that accepts basically the same types of SQL statements, not all databases have the same components. An Oracle database is considerably more complicated than some other PC-based databases you may have seen, such as Microsoft Access or even SQL Server. The components of an Oracle database are broken into three basic areas, corresponding to the three basic areas of host machines that run Oracle databases. In this section, pay close attention to how each component in each part of the Oracle database interacts with a component in another part. Figure 1-1 illustrates the various elements of the Oracle database, which are tested thoroughly on OCP Database Administration Fundamentals I exam (1Z0-022). The components are as follows:

- **Memory**  The Oracle System Global Area (SGA), sometimes also called the Shared Global area because this allocation of memory is shared between all users of the Oracle database.

- **Disk**  Oracle datafiles, redo logs, control files, password files, and parameter files. These files contain the stored data of the Oracle database.

- **Processes**  Threads in the oracle.exe background process (Windows) or individual processes (UNIX) and the server process. These processes do all the behind-the-scenes work to keep the Oracle database functioning properly.

**TABLE 1-1. Comparing Relational Databases to Other Data Management Systems**

<table>
<thead>
<tr>
<th>Task</th>
<th>FlatFile System</th>
<th>RDBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handles parent/child data relations?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Handles other types of data relationships?</td>
<td>Not well</td>
<td>Yes</td>
</tr>
<tr>
<td>Handles data manipulation easily?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Easily adaptable to changing business needs?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Handles data retrieval easily?</td>
<td>Sometimes</td>
<td>Yes</td>
</tr>
<tr>
<td>Handles data retrieval quickly?</td>
<td>Sometimes</td>
<td>Sometimes</td>
</tr>
</tbody>
</table>
FIGURE 1-1. Oracle server architecture
Oracle SGA
Oracle’s memory component, the SGA, consists of several elements, each of which is designed for a specific purpose.

Buffer Cache  The buffer cache stores Oracle data in memory for users to view or change. In this way, users never make changes directly to disk files. Instead, Oracle reads the appropriate data into memory for the user process to change, and it writes the changes back to disk at some point later. The buffer cache follows a modified least-recently used (LRU) algorithm to determine when data in this area can be eliminated when more space is needed in the buffer cache to make room for user data requested. The information in the buffer cache is shared among all concurrent users connected to the Oracle database.

Log Buffer  The log buffer stores special information called redo, which helps Oracle reconstruct data changes in the event of a system failure. Redo information is written to the log buffer by users making data changes and is stored in the log buffer until Oracle can write the redo information to disk.

Shared Pool  The shared pool stores many items that are “mission critical” to the operation of your Oracle database. Components of the shared pool include the library cache, for storing parsed SQL statements for reuse by other users; the dictionary or row cache, for storing Oracle data dictionary information in memory where it can be accessed quickly; and latches and other database-control mechanisms.

TIP
The Oracle data dictionary is a set of information stored in Oracle that tells you all kinds of important things about your database. The data dictionary is used frequently by users and Oracle processes alike, so it is important for overall database performance to store dictionary information in memory whenever possible. Hence, you can see the need for the dictionary cache in your shared pool.

Large Pool  The fourth and less frequently used component of Oracle’s SGA is the large pool, which is a large memory allocation area used to support Oracle backup and restore operations, I/O server processes, and session memory for the shared server. Introduced in Oracle8, this component is optional for Oracle database operation.
Other Memory Areas  There are other components to the SGA in Oracle8i and later versions, such as the Java pool and large pool, that are not shown in Figure 1-1. The items not included in this discussion and/or the figure are excluded because a detailed examination of these topics is not necessary for passing the OCP exam on SQL. This figure and the current discussion are merely meant to give you the larger picture of Oracle before digging into a meaningful discussion of SQL queries.

Oracle Disk Components
The Oracle disk components store all kinds of vital information in your Oracle database. You cannot run Oracle without having all your disk components (except password files) in their proper places.

Datafiles  This mandatory disk component is used for storing Oracle dictionary and application database objects. These components often grow extremely large in size. Information in the buffer cache and the dictionary cache in memory comes from datafiles on disk. Every Oracle database has at least one datafile (but usually more). Datafiles store Oracle data. If you create a table in Oracle and populate it with rows, Oracle places the table and the rows in a datafile. Each datafile can be associated with only one database.

Redo Logs  This mandatory disk component is used for storing redo information on disk. Information from the log buffer in memory eventually gets written here. This component is meant to record all changes made to data in your Oracle database. These logs are critical for recovery of data in the event of a database failure.

Control Files  This mandatory disk component is used for storing vital information about the location of Oracle disk components on the host system. The physical locations of both datafiles and redo logs in the server’s file system are stored in your control file. There can be one or many control files in an Oracle database. If there is more than one control file, each will be an identical copy. Oracle reads the control files every time you start the database and updates the control files when redo logs or datafiles are added or moved.

Password Files  This optional disk component is used for securing privileged user connection information to allow the database to be managed remotely via Enterprise Manager, Oracle’s database-management tool. It also controls the number of the privileged system-management connections that can be made to the database at the same time. Without a password file, you may only administer your database by connecting directly to the machine hosting the Oracle database and using management tools such as SQL*Plus directly from the host machine.
**Parameter Files**  This mandatory disk component is used for configuring how Oracle operates while it is running. To start the database instance, Oracle must read the parameter file to determine what the configuration parameters are for that instance. A parameter file contains many parameters and their set values. Oracle reads the parameter file when you start the database. Some Oracle professionals refer to the parameter file as the *init.ora* file. You may maintain one or many parameter files for a database, corresponding to different instance configurations you may want to implement at various times.

**Oracle Server and Background Processes**
The final component of Oracle to be covered is the set of elements that comprise Oracle on your host system’s CPU. The Oracle server process reads data from datafiles into the buffer cache on behalf of user processes. It can either be shared between multiple users or be dedicated to one user. The Oracle database also has one background process in Windows environments—*oracle.exe*. If you hit **CTRL-ALT-DELETE** on your system hosting the Oracle database, click the Task Manager button to bring up the Task Manager, and then click on the Processes tab, you will see this process running on your Windows machine. In Windows, this process has many threads that handle other important activities your database is engaged in at all times in the background. If you want to find information in Windows about services setup for use with Oracle software, you can look in Start | Settings | Control Panel. For NT, the Services icon lists all the Windows services available on the machine. For Windows 2000, you can double-click the Administrative Tools icon to find the Services icon. On UNIX machines, Oracle consists of multiple background processes. If the database is running on a UNIX machine, you can usually see its background processes if you issue the command `ps -fu oracle` on your UNIX command line.

**What an ORDBMS Is**
As object-oriented programming has gained popularity, Oracle has adjusted its relational database-management paradigm to include support for object-relational database design. This methodology incorporates the best features of object programming with the best features of relational programming and allows the developer to draw from both when designing a system in Oracle. Two important features supported on the object side include:

- Permitted users to define the structure of the data they wish to store
- Allowing users to define programmatic methods for manipulating that data and associating those methods directly to the data stored
For Review
Know the three components of the Oracle database, and be able to name each of
the elements in each component.

Exercises
1. You are examining the components of an Oracle database. Which of the following choices identifies an aspect of Oracle that resides on the disk of
the machine hosting the Oracle database?
   A. SGA
   B. Datafile
   C. Background process
   D. Java pool

2. You are interested in seeing Oracle running on your Windows-based host
machine. In which of the following areas would you look?
   A. Control Panel | Services Icon
   B. Desktop
   C. Windows Explorer
   D. Start Menu

3. You are interested in seeing Oracle running on your UNIX machine. Which
of the following commands might you use?
   A. ls
   B. grep
   C. ps
   D. df

Answer Key
1. B. 2. A. 3. C.
Usage and Benefits of PL/SQL

How to write programs in PL/SQL is no longer a topic being tested on the OCP Introduction to SQL exam. Nevertheless, it is worth your time as an Oracle professional to know about the existence of PL/SQL and its usage and benefits. PL/SQL is Oracle's own language for developing database applications. In addition to supporting all SQL operations that Oracle SQL supports, PL/SQL adds programming language extensions such as conditional statement processing, loops, variables, cursor operations, abstract datatypes, modularization, encapsulation, overloading, and more. The following bullets list frequently cited reasons why PL/SQL developers use the language:

- **PL/SQL is easy to learn and use.** Professionals with even a modest programming background can usually pick up PL/SQL syntax before too long and develop programs of moderate complexity without much effort. Professionals without a programming background can learn PL/SQL with more effort spent learning basic constructs, such as variable declaration, conditional statement processing, and so on.

- **PL/SQL is stored in the Oracle database, dramatically improving performance.** This means that you only have to compile the code into the Oracle database to make that code available to every user on the system. There is no need for an extended deployment as with traditional client/server applications. The result is code that runs quickly and works natively with your Oracle data.

- **PL/SQL integrates well with the Oracle database.** No special command syntax is needed to perform SQL operations involving data in the Oracle database. No colons, question marks, or other odd characters are required to prefix variables as in other languages. One exception to this rule relates to trigger development, which is a hybrid between a database object and PL/SQL.

- **PL/SQL is especially adept at processing large blocks of data.** Oracle PL/SQL provides a special construct called a cursor for loop, which allows you to query several rows of table data and then process through each row of that data in an iterative fashion. This feature allows you to process large amounts of data in bulk.

- **PL/SQL comes with lots of Oracle-supplied code to assist in performing tasks.** Oracle distributes several packages of PL/SQL code with every database shipped. This code enables you to perform highly specialized operations, such as file input/output, or I/O, retrieving Web pages into your database, job scheduling, dynamic SQL, interprocess communication,
resource management, and much more. You can refer to these Oracle-supplied packages just like any other PL/SQL program.

- **PL/SQL supports named and anonymous programs.** There are many different types of named programs you can develop in PL/SQL, including stored procedures, functions, and packages. These code blocks are actually compiled and stored in the database and are available for later use. You can also write anonymous programs, which are compiled at the time you submit the code for execution, executed, but not stored in the database.

- **PL/SQL can be integrated into database tables via triggers.** Oracle integrates PL/SQL programmatic activity into database tables via triggers. This feature allows you to develop applications that use complex business rules for regulating data inside the database, thus reducing the potential for corrupt or inappropriate data from users.

- **PL/SQL supports encapsulation and modularization.** Encapsulation involves using one named PL/SQL program to call another named PL/SQL program. Modularization involves breaking down a large task into several smaller components and then writing named PL/SQL programs to handle those smaller tasks. The result is code that's easier to read and maintain.

- **PL/SQL supports overloading.** Overloading occurs when you have a package containing procedures or functions with the same name that accept different variables of different datatypes. When you call the overloaded procedure, Oracle dynamically decides which version of the procedure to use based on the datatype of the variable you pass.

- **PL/SQL allows programmers to package their Oracle code.** Oracle PL/SQL supports a construct called a package. This feature allows you to logically group several procedures or functions that work together into one single construct. Procedures grouped together using packages perform better than they would individually because all procedures in the package will be loaded into memory as soon as one of the procedures is referenced. In contrast, stand-alone procedures are only loaded into memory when called. This reduces the overhead Oracle requires for memory management, thus improving performance.

- **PL/SQL supports advanced datatypes.** PL/SQL gives users the ability to define abstract datatypes, such as records, allowing you some object-oriented flexibility in your procedural code. PL/SQL also offers table constructs for variable definition and use, approximating the use of arrays. Finally, PL/SQL allows you to declare REF datatypes, which gives PL/SQL the ability to use datatypes similar to pointers in C and C++.
PL/SQL code is portable. You can write a PL/SQL program on an Oracle database running on Solaris and then move the program to Oracle running on Windows 2000 or some other operating system without rewriting the program.

For Review
Be sure you understand the benefits of PL/SQL programming at a conceptual level as part of the Oracle database. However, OCP Exam 1 no longer tests your knowledge of how to develop PL/SQL applications, so a conceptual level is sufficient at this point.

Exercises

1. You develop a PL/SQL package for use with Oracle. Which of the following choices identifies where that code is stored?
   A. As an executable file on the host system
   B. As uncompiled code in the database
   C. As compiled code in the database
   D. As a flat file, sent to the database when you want to run the program

2. You want to develop a PL/SQL package containing different procedures with the same name but different variable datatypes. What is the name of the PL/SQL feature that allows this?
   A. Packaging.
   B. Overloading.
   C. Encapsulation.
   D. This functionality is not possible in PL/SQL.

3. What is the name of the special loop that makes PL/SQL especially adept at processing large numbers of data records? ________________

Answer Key
1. C. 2. B. 3. The cursor for loop.
Writing Basic SQL Statements
This section will cover the following areas related to selecting rows:

- Capabilities of SQL `select` statements
- Executing `select` statements
- Differentiating between SQL and SQL*Plus commands

Now, let's dig in and start your approach to Oracle systems. This section maps directly to objectives on the OCP exam, and in the section you will learn what SQL provides you in the Oracle working environment. You'll also cover how to develop the all-important `select` statement, used for obtaining data from Oracle. You will even learn how to distinguish SQL commands from SQL*Plus commands. This skill becomes increasingly important as you use SQL*Plus for developing and running queries and because there are certain SQL*Plus commands you must know for passing the OCP exam.

Capabilities of SQL `select` Statements
If you've already developed SQL code for other database applications, you're in for some good news. Oracle SQL complies with the industry-accepted standards, such as ANSI SQL92. But before exploring SQL `select` statements in detail, consider the following overview of all the statement categories available in SQL and their associated usage:

- `select` Used for data retrieval and query access. Many developers consider this statement to be part of data manipulation language (DML) operations against the database. However, Oracle does not. When OCP refers to DML statements, you should make a mental note that Oracle is not referring to the `select` command.

- `insert, update, delete` Used for DML operations against the Oracle database, including adding new records, changing existing records, and removing records, respectively.

- `create, alter, drop` Used for data definition language (DDL) operations against the Oracle database, including adding, modifying, and removing database objects such as tables, indexes, sequences, and so on, respectively.

- `commit, rollback, savepoint` Used for transaction-control activities inside a user's session, including saving changes, discarding changes, and marking logical breakpoints within the transaction, respectively.
grant, revoke  Used for data control language (DCL) operations against your Oracle database, where you might need to control user access to data.

**Getting Started: SQL*Plus**

Many developers, designers, DBAs, and power users begin their experience with Oracle using an existing Oracle application in an organization. The first tool many people see for selecting data directly from the Oracle relational database management system is SQL*Plus. When users first start SQL*Plus, in most cases, they must enter their Oracle username and password in order to begin a session with the Oracle database. There are some exceptions to this rule that use the password authentication provided with the operating system. The following example shows how you might begin a session with Oracle on the UNIX command line if the database is present on the UNIX machine you are connected to:

```
$ /home/oracle> sqlplus scott/tiger
```

**TIP**

From Windows, you can execute the above command at a DOS prompt to run the command-line version of SQL*Plus. Or, you can click on Start | Programs | Oracle ORACLE_HOME | Application Development | SQL*Plus to run the GUI version of SQL*Plus. On most systems, ORACLE_HOME will be replaced with the name of the Oracle software home location, such as OraHome1.

Alternately, if you want to connect to an Oracle database not present on the machine you are currently connected to, you might issue the `sqlplus` command with a specified database name tacked onto the end of your username and password, as you'll see in the code block following this paragraph. That extra `@orcl` tacked onto the end of your username and password tells the operating system the name of the Oracle database you want to connect to. Here's the example:

```
$ /home/oracle> sqlplus scott/tiger@orcl
```

**TIP**

For our purposes in this book, we'll assume that the Oracle database you want to connect to is present on the same machine where you'll be running SQL*Plus.
Whenever you log into Oracle via SQL*Plus, you create a session with the database. A session is an interactive runtime environment similar to a command-line environment, such as UNIX or DOS, in which you enter commands to retrieve data. Oracle performs a series of activities to obtain the data you ask for based on the SQL command you enter. The session starts as soon as you log into Oracle, and ends when you log out. Think of it as a conversation, which in turn implies language. Remember, you communicate with Oracle using the structured query language, SQL, to obtain the information you need.

**TIP**
To connect to the database, you must be granted permission to do so—simply having a user ID and password isn’t enough. For more information on permissions, see Chapter 8.

SQL is a functional programming language, which means that you specify the types of things you want to see happen in terms of the results you want. You define the result you want, and Oracle determines how to get it for you. Take another look at the select statement I showed you earlier:

```
SQL> SELECT EMPNO, ENAME, SAL FROM EMP
     2  WHERE ENAME = 'SMITH';
```

The first point you should understand about SQL statements is that they can be entered across multiple lines. Our statement above contains two lines of keywords and text string expressions. However, notice also that we did not split any keywords across two lines—this is not permitted in Oracle. Finally, SQL statements are not case-sensitive. Thus, the following statement is logically equivalent to the one shown above:

```
SQL> select empno, ename, sal from emp
     2  where ename = 'SMITH';
```

**NOTE**
While column names, table names, and keywords (such as select, from, and where) are not case-sensitive, text strings like SMITH, appearing in the code sample above in single-quotes, are case-sensitive. This is because Oracle stores the text exactly as you type it, so if you typed SMITH in uppercase when you stored that string in the EMP table, then that is exactly what Oracle stored.
Sometimes text strings are called literals for this reason—they are literally what you entered.

Now let’s look at the content of the SQL statement. This statement asks Oracle to provide data from the EMP table, where the value in a certain column called ENAME equals SMITH. We don’t care how Oracle gets it, just as long as Oracle returns only the record from table EMP we asked for. Contrast this approach to other languages you may have heard about or programmed in, such as C++ and COBOL. These languages are often referred to as procedural or iterative programming languages because the code written in these languages implies an end result by explicitly defining the process for obtaining the result. The following block of code from an imaginary procedural programming language similar to C illustrates how the same function may be handled by explicitly defining the means to the end:

```c
#include <stdio.h>
#include <string.h>
#include <rdbms.h>

int *empno;
char *statement;

typedef emp_rec record {
  int empno;
  char[10] emp_name;
  int sal;
};

void main() {
  login_to_oracle(scott,tiger);
  access_table(emp);
  open(statement.memaddr);
  strcpy("SELECT EMPNO, ENAME, SAL FROM EMP WHERE ENAME = 'SMITH'", statement.text);
  parse(statement);
  execute(statement);
  for (i=1, i=statement.results, i+1)
    fetch(statement.result[i], emp_rec);
    printf(emp_rec);
  close(statement.memaddr);
}
```

Of course, this C-like block of code will not compile anywhere but in your imagination, but the point of the example is clear—other languages make you define the process, whereas SQL lets you define the result.
For Review
What is SQL? What is SQL capable of? How does SQL compare to other programming languages you might use, such as Java and C?

Exercises

1. You are determining which type of SQL statement to use in your Oracle database. Which of the following choices identifies the type of statement you would use when trying to obtain data from the database?
   A. select
   B. update
   C. insert
   D. delete

2. Which of the following choices identifies a functional programming language?
   A. C
   B. Java
   C. COBOL
   D. SQL

3. Identify a command that is part of SQL’s data control language (DCL).
   ________________________

4. Identify a command that is part of SQL’s data manipulation language (DML). ________________________

5. Identify a command that is part of SQL’s data definition language (DDL).
   ________________________

Answer Key
1. A, 2. D, 3. B. grant or revoke 4. select, update, delete, or insert Oracle considers the select command to be part of data manipulation language even though it technically doesn’t allow you to change the data being stored. 5. create, alter, drop
Executing select Statements

The most common type of SQL statement executed in most database environments is the select statement, which queries a table in the database for requested data.

Tables in Oracle are similar in concept to spreadsheets (but not necessarily similar to the table in your kitchen!). Examine the following code block, where you see a select statement in the context of a session with Oracle:

$HOME/oracle> sqlplus scott/tiger
SQL*Plus: Release 8.1.7.0.0 - Production on Fri July 06 18:53:11 2001
Copyright (c) Oracle Corporation 1979, 2000. All rights reserved.
Connected to: Oracle9i Release 9.0.1.0.0
With the partitioning option
JServer Release 9.0.1.0.0 Production
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          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.

TIP
The last part of the code block is where Oracle tells you how many rows it obtained from the database table in response to your SQL command. Later code blocks in this book omit that part to conserve space. You can tell SQL*Plus not to display row count information as well using the set feedback off command. We’ll talk more about SQL*Plus commands later in the chapter.
The first part, containing the copyright information, is a welcome message from SQL*Plus. If you wanted, you could suppress this information in your call to SQL*Plus from the operating system command line by entering `sqlplus -s` and pressing ENTER, where the `-s` extension indicates SQL*Plus should run in silent mode. This is sometimes useful for batch programs that write output to an automated feed file where you don’t want a lot of extraneous junk in the feed because an error will result. We’ll explore some other SQL*Plus commands that help you control the appearance of your output later in the chapter. The bold line in the block illustrates a simple SQL `select` statement. In essence, you’re asking Oracle to return all data from all columns in the EMP table. Oracle replies with the contents of the EMP table. The main components of a `select` statement are listed next, and both are required in every `select` statement you issue on the database:

- **The `select`, or `column`, clause** This clause contains columns or expressions containing data you want to see, separated by commas. The preceding query uses a wildcard (*) character, indicating we want data from every column in the table.

- **The `from`, or `table`, clause** This clause tells Oracle what table to get the data from.

**TIP**
Always use a semicolon (;) to end SQL statements when entering them directly into SQL*Plus. You can use a slash (/) in some situations, such as for SQL*Plus batch scripts, but be careful—a slash at the end of a SQL statement already ended with a semicolon makes the statement run twice!

**A Note about Columns and Datatypes**
Tables in the Oracle database are comprised of columns, each storing a unit of information for the row. These units taken together across a single row comprise a record stored in the table. Review the first record in the preceding code block for EMPNO 7369, which is listed here:

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>CLERK</td>
<td>7902</td>
<td>17-DEC-80</td>
<td>800</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Each column identifies an aspect of this unique employee. EMPNO identifies his employee number, ENAME identifies his name, and so on. The information
stored in each column of the table for this fellow must correspond to the datatype defined for that column. For example, column EMPNO is defined as a NUMBER column, meaning that only numbers of a certain size can be stored for records in that column. No text, date, or nonnumerical information can be stored in EMPNO, because doing so would violate the column’s stated datatype. I’ll refer to a column’s datatype frequently throughout the rest of the book, so it’s worth your time to master this fundamental concept. The column datatypes permitted in Oracle tables that we’ll work with most frequently are listed here:

- **NUMBER** A datatype used for storing numerical data. No dashes, text, or other nonnumerical information are allowed in columns of this datatype.

- **DATE** A datatype used for storing date information. Internally, Oracle stores dates as numbers, which it can then convert into any DATE format you want. By default, DATE information is displayed in DD-MON-YY format (for example, 25-DEC-79).

- **VARCHAR2** A datatype used for storing text data. Any text character (including special characters, numbers, dashes, and so on) can be stored in a VARCHAR2 column.

- **CHAR** A datatype used for storing text data. Any text character (including special characters, numbers, dashes, and so on) can be stored in a CHAR column, padded with spaces so that the text stored in the CHAR column fills the entire length available. Thus, the name SMITH would take up ten spaces in a column declared as CHAR(10), even though the name itself is only five characters long, with the other five characters occupied by blank spaces.

**TIP**

The main difference between VARCHAR2 and CHAR columns is the amount of space required for storing text data, which is greater for CHAR columns than for VARCHAR2 columns. This is because CHAR columns have a fixed length and always store the same number of bytes, whereas VARCHAR2 has a variable length and only contains the number of bytes you provide it.

Datatypes for storing other types of information exist in Oracle; however, there aren’t as many of them as you might encounter in database products from other vendors. For example, Oracle has no currency datatype. Monetary values are treated simply as numbers, and as such they can be stored in a column defined as the NUMBER datatype.
**TIP**

Another datatype we’ll observe from time to time in the book is the ROWID datatype. This is a special datatype used by Oracle to format the information used to display the physical location of the row on disk.

---

**The “Schema” of Things**

Take a look at the following code block:

```sql
SQL> select empno, ename, sal
2  from scott.emp;
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>800</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>1600</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>1250</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>2975</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>1250</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>2850</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>2450</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>3000</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>5000</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>1500</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>1100</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>950</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>3000</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>1300</td>
</tr>
</tbody>
</table>

Notice anything different about the way table EMP is referenced in this table clause? It has the name of the owner, SCOTT, prefixed to it. Oracle developers and DBAs refer to the concept of referencing the table owner as well as the table itself as a schema. If you create a database object such as a table, this object belongs to you. It is part of your schema. The identity you use when you log into your database to run `demobld.sql` determines the schema that all those tables will belong to.

When the table you reference in a query isn’t prefixed with the schema it belongs to, Oracle assumes the table exists in your schema and tries to query it. If the table doesn’t exist in your schema, you must prefix the table name with the schema information, separating the schema owner from the table name with a period.
TIP

A schema is a logical grouping of database objects based on the user who owns the objects.

Prefixing Columns with Table Names
The same aliasing concept works in the column clause, too—you can prefix the column name with the table name separated by a dot (.) in the table clause for your query. Make sure you understand how to specify a schema owner, the table name, and the column name in a select statement in SQL*Plus. The following code block demonstrates the most formal usage for prefixing with appropriate schema and table information:

```sql
SELECT table_name.column_name, table_name.column_name
FROM schema.table_name;
```

Arithmetic and Table Data
Oracle lets you perform arithmetic operations on your numeric table data as well. The operators used in Oracle are the same as in daily use (+ for addition, − for subtraction, * for multiplication, and / for division). Say, for example, you are performing a simple annual review that involves giving each user a cost-of-living increase in the amount of 8 percent of his or her salary. The process involves multiplying each person's salary by 1.08. Oracle makes the work easy if you use arithmetic expressions, as shown here:

```sql
SQL> select empno, ename, sal, sal*1.08
  2  from emp;
EMPNO ENAME     SAL SAL*1.08
--------- ---------- --------- ---------
7369 SMITH 800 864
7499 ALLEN 1600 1728
7521 WARD 1250 1350
7566 JONES 2975 3213
7654 MARTIN 1250 1350
7698 BLAKE 2850 3078
7782 CLARK 2450 2646
7788 SCOTT 3000 3240
7839 KING 5000 5400
7844 TURNER 1500 1620
7876 ADAMS 1100 1188
7900 JAMES 950 1026
7902 FORD 3000 3240
7934 MILLER 1300 1404
```
Operator Precedence
There’s usually at least one question on OCP dealing with operator precedence—that high-school math concept regarding which calculation to do first. An easy way to remember operator precedence in mathematics is to use the acronym PEMDAS. You can remember PEMDAS using the mnemonic “Please Excuse My Dear Aunt Sally.” PEMDAS stands for parentheses, exponents, multiplication and division, addition and subtraction. Here are some examples of PEMDAS in action:

- \( 2 + 6 / 2 \) equals 5
- \( (2 + 6) / 2 \) equals 4
- \( 2 / 10 + 36 * (84 - 6) \) is 2808.2
- \( 2 / 10 + 36 * 84 - 6 \) is 3018.2.

2 + 2 and the DUAL Table
As mentioned earlier, every select statement must have a column clause and a table clause. However, you might not always want to perform arithmetic calculations on data from an actual table. Say, for example, you simply want to add 2 + 2. Conveniently, the column clause in a select statement needn’t contain actual column names. It can contain fixed numbers or other types of expressions instead. But what about the table clause? Because you’re using fixed numbers, you don’t want data from a real table. So why not use a fake one? You can use a special table called DUAL to fill in the table clause without Oracle actually using its data.

Take a look at the following block:

```sql
SQL> select 2 + 2 from dual;
 2+2
-------
   4
```

The DUAL table consists of one column, called DUMMY, containing one value, X. Execute a `select * from DUAL` statement and see for yourself that there is no meaningful data stored here. It simply exists as a SQL construct to support the requirement of a table specification in the `from` clause. The DUAL table is owned by the Oracle built-in user SYS. We can also use the DUAL table in our understanding of schemas. The following example shows you how to obtain the username you used when you logged into Oracle:

```sql
SQL> select user from dual;
USER
-----
SCOTT
```
Handling NULL Values
Sometimes a query for information produces a nothing result. In database terms, nothing is called NULL. In set theory, the mathematical foundation for relational databases, NULL represents the value of an empty dataset, or a dataset containing no values. Put another way, NULL is not the blank character displayed when you hit the spacebar, nor is it somehow equivalent to zero! NULL is the absence of information. Unless specified otherwise, a column in a table is designed to accommodate the placement of nothing into the column. An example of retrieving NULL is listed in the MGR column of the following code block on EMPNO 7839:

```
SQL> select empno, ename, mgr
  2  from emp;
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>MGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>7902</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>7698</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>7698</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>7839</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>7698</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>7839</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>7839</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>7566</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td></td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>7698</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>7788</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>7698</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>7566</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>7782</td>
</tr>
</tbody>
</table>

However, there are times when you may want to substitute a value in place of NULL. Oracle provides this functionality with a special function, called NVL(). Assume that you do not want to see blank spaces for manager information. Instead, you want the output of the query to contain a zero where a NULL value is listed. The query in the following code block illustrates how you can obtain the desired result:

```
SQL> select empno, ename, nvl(mgr,0)
  2  from emp;
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>NVL(MGR,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>7902</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>7698</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>7698</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>7839</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>7698</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>7839</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>7839</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>7566</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>7698</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>7698</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>7788</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>7698</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>7566</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>7782</td>
</tr>
</tbody>
</table>
The basic syntax for nvl() is NVL(column_name, value_if_null). Notice that the column specified in nvl() contains an actual value. That value is what Oracle returns; when the column is NULL, the special string is returned. The nvl() function can be used on columns of all datatypes, but remember this: The value specified to be returned if the column value is NULL must be the same datatype as the column specified.

The distinct Keyword
If you look back at the code block that lists all the employees in the EMP table, you’ll notice something interesting in the JOB column. Many of the employees have the same job title. Sometimes, you might have a situation where you want to see only the unique values for a column that you know contains many repeated values. In order to do so, Oracle offers the distinct keyword. To obtain the unique values for a column containing duplicates, you simply precede the column reference with the distinct keyword in your column clause, like this:

```
SQL> select distinct job
    2  from emp;
```

TIP
In order for the distinct keyword to work, it must appear directly after the select keyword in your SQL query.
When more than one column name appears after the distinct keyword in a select statement, then Oracle attempts to identify all the distinct combinations of values in those columns named. Take a look at an example:

```
SQL> select distinct job, empno from emp;
JOB  EMPNO
--------- ---------
ANALYST    7788
ANALYST    7902
CLERK      7369
CLERK      7876
CLERK      7900
CLERK      7934
MANAGER    7566
MANAGER    7698
MANAGER    7782
PRESIDENT  7839
SALESMAN   7499
SALESMAN   7521
SALESMAN   7654
SALESMAN   7844
```

Changing Output Headings with Aliases

In every result set Oracle returns in response to your SQL select commands, Oracle creates headings for each column so that you know what the data is. By default, Oracle reprints the column name exactly as you defined it in the select statement, including functions if there are any. Unfortunately, this method often leaves you with a bad description of the column data. Oracle truncates the expression to fit a certain width corresponding to the datatype of the column returned, making the problem even worse. Fortunately, you can use aliases in your column clause to solve this problem. In a column alias, you give the column another name that Oracle uses when the select statement results are displayed. This feature gives you the ability to fit more descriptive names into the space allotted. Here’s an example:

```
SQL> select empno, ename, nvl(mgr,0) as mgr
2   from emp;
EMPNO ENAME   MGR
--------- ---------- ---------
7369  SMITH   7902
7499  ALLEN  7698
7521  WARD   7698
7566  JONES  7839
7654  MARTIN 7698
```
TIP
You can omit the as keyword in the column alias and still wind up with substantially the same result.

Column aliases are useful for adding meaningful headings to output from SQL queries. Aliases can be specified in two ways: either by naming the alias after the column specification separated by a space or by using of the as keyword to mark the alias more clearly. Here’s the general rule:

```
SQL> -- SELECT column_name_or_operation alias, ...;
SQL> SELECT nvl(mgr,0) MGR
2  FROM EMP;
```

or

```
SQL> -- SELECT column_name_or_operation AS alias, ...;
SQL> SELECT nvl(mgr,0) AS MGR
2  FROM EMP;
```

You don’t need to specify a function in order to use an alias. For example, if you simply wanted to change the column heading for the MGR column to something more descriptive, you could do so using a column alias. The SQL statement might look something like `SELECT mgr as "Manager Code" from emp`.

**Putting Columns Together with Concatenation**
You can also glue together column data to produce more interesting or readable output. This is called *concatenation*. The concatenation operator is two pipe characters put together: `||`. You can also use the `concat( )` operation, passing it the two column names. In the following example, the ENAME column is concatenated with a text expression and the JOB column using both available methods to produce a meaningful result:

```
SQL> select ename ' , who is the ' ||
2  concat(job,' for the company')
```
```sql
3  as "Name and Role"
4  from emp;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>CLERK</td>
</tr>
<tr>
<td>ALLEN</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>WARD</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>JONES</td>
<td>MANAGER</td>
</tr>
<tr>
<td>MARTIN</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>BLAKE</td>
<td>MANAGER</td>
</tr>
<tr>
<td>CLARK</td>
<td>MANAGER</td>
</tr>
<tr>
<td>SCOTT</td>
<td>ANALYST</td>
</tr>
<tr>
<td>KING</td>
<td>PRESIDENT</td>
</tr>
<tr>
<td>TURNER</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>ADAMS</td>
<td>CLERK</td>
</tr>
<tr>
<td>JAMES</td>
<td>CLERK</td>
</tr>
<tr>
<td>FORD</td>
<td>ANALYST</td>
</tr>
<tr>
<td>MILLER</td>
<td>CLERK</td>
</tr>
</tbody>
</table>

**TIP**

*Use column aliases to name your concatenated column to make the output more readable and meaningful.*

**For Review**

1. Understand the two components of `SELECT` statements and what a schema is.

2. Know how to perform arithmetic on selected columns and on numeric expressions in Oracle and know what the DUAL table is.

3. Know both methods used for concatenating columns and how to define column aliases. Also, know what the `DISTINCT` keyword is and how it is used.

4. Be able to define what NULL means in the context of Oracle SQL and how to use the `NVL()` function.

5. Be sure you understand the correct operator precedence using the acronym PEMDAS.
Exercises

1. You are identifying a table for use in your `select` clause that was not created by you. Which of the following choices identifies the reference that must be included in your `select` statement so that Oracle knows where to look for the information?

   A. Alias
   B. Schema
   C. Expression
   D. Session

2. Use the following code block to answer this question:

   ```sql
   SQL> select empno, ename, mgr
           2  from emp;
   ---------- ------- ---------
   EMPNO ENAME MGR
   -------- -------- --------
   7369 SMITH 7902
   7499 ALLEN 7698
   7521 WARD 7698
   7566 JONES 7839
   7654 MARTIN 7698
   7698 BLAKE 7839
   7782 CLARK 7839
   7788 SCOTT 7566
   7839 KING
   7844 TURNER 7698
   7876 ADAMS 7788
   7900 JAMES 7698
   7902 FORD 7566
   7934 MILLER 7782
   SQL> select empno, ename, nvl(mgr,'none') as mgr
           2  from emp;
   ---------- --------------
   EMPNO ENAME MGR
   -------- -------- --------
   7369 SMITH 7902
   7499 ALLEN 7698
   7521 WARD 7698
   7566 JONES 7839
   7654 MARTIN 7698
   7698 BLAKE 7839
   7782 CLARK 7839
   7788 SCOTT 7566
   7839 KING
   7844 TURNER 7698
   7876 ADAMS 7788
   7900 JAMES 7698
   7902 FORD 7566
   7934 MILLER 7782
   SQL> select empno, ename, nvl(mgr,'none') as mgr
           2  from emp;
   Which of the following choices describes what Oracle will return as the output in the MGR column for KING's record from this query?
   A. Oracle returns NULL in the MGR column for KING's record.
   B. Oracle returns MGR in the MGR column for KING's record.
   C. Oracle returns NONE in the MGR column for KING's record.
   D. Oracle returns an error.
3. You are concatenating information from two columns in an SQL query. Which of the following choices best identifies the special character required for this operation?
   A. @
   B. #
   C. ||
   D. /

4. Provide the name of the table containing no meaningful information that can be used to fulfill the table clause requirement for select statements when you perform arithmetic operations on fixed numeric expressions:

5. You may use the contents from the standard EMP table used in this discussion to answer the following question. You are attempting to calculate 20 percent of the salary and commission for all employees of the company. Which of the following SQL statements would be appropriate for the task?
   A. select empno, ename, sal/20, comm/20 from emp;
   B. select empno, ename, sal*20, comm*20 from emp;
   C. select empno, ename, sal/.20, comm/.20 from emp;
   D. select empno, ename, sal*.20, comm*.20 from emp;

6. You may use the contents of the following code block to answer this question:

   SQL> select * from dept;
   DEPTNO DNAME          LOC
   --------- -------------- -------------
   10 ACCOUNTING     NEW YORK
   20 RESEARCH       DALLAS
   30 SALES          CHICAGO
   40 OPERATIONS     BOSTON

   You issue the following statement in Oracle: select distinct dname, loc from dept. Which of the following choices correctly describes the result Oracle will return?
A. Oracle returns the distinct combinations of values from DNAME and LOC.

B. Oracle returns only three distinct values from DNAME in the DEPT table.

C. Oracle returns only the distinct values from the DEPTNO column.

D. Oracle returns the contents of all four records from the table.

**Answer Key**

1. B. 2. D. Remember, the datatype in the `nvl( )` function must match the datatype for the column. 3. C. 4. DUAL. 5. D. 6. A.

**Differentiating Between SQL and SQL*Plus Commands**

Although the SQL*Plus work environment works well when you don’t make mistakes, it is unforgiving to the fat-fingered once you have pressed `ENTER` to move to the next input line. So far, this limitation hasn’t presented much difficulty because our queries haven’t been long. However, as the queries you write get more and more complicated, you will grow frustrated. SQL*Plus does allow some correction of entered statements with a special command called change, abbreviated as `c`.

Consider the following example, which illustrates this point:

```
SQL> SELECT empno, ename, NVL(mgr,'none') mgr,
       2 hirdate, sal, comm, deptno
       3 FROM EMP;
SELECT empno, ename, NVL(mgr,'none') mgr,
FROM EMP;
ERROR at line 1:
ORA-01722: invalid number
SQL> /1*
1* SELECT empno, ename, NVL(mgr,0) mgr,
SQL> c/'none'/0
1* SELECT empno, ename, NVL(mgr,0) mgr,
SQL> /
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
```

---

**OCP Introduction to Oracle9: SQL Exam Guide**

38
In this example, the select statement contains a datatype mismatch error in the nvl( ) function. Oracle notices the error and alerts you to it with the ORA-01722 error message.

Other error messages that may be produced include the following:

ORA-00904: invalid column name

This error indicates that the column you referenced does not exist or was misspelled or misplaced. To resolve this problem, you need to check for typos in your column clause and verify that the column actually exists in the table. Sometimes, a column name may include nonalphanumeric characters, such as underscores, designed to separate two words. Thus, the column name EMPNO is not the same as EMP_NO, even though conceptually they mean about the same thing.

ORA-00923: FROM keyword not found where expected

This error indicates that the FROM keyword was not included or was misspelled. Sometimes this error occurs when you put a comma after the last column listed in your select clause (that is, select empno, ename, from emp), so watch out for that common mistake.

ORA-00942: table or view does not exist

This error indicates that the table or view typed in does not exist. Usually, the ORA-00942 error message indicates a typo in the name of the table or view, or that the schema owner was not specified in front of the table name. This error is fixed either by correcting the typing problem or by adding the schema owner onto the front of the table name. (An alternative solution for the latter case involves creating synonyms for tables that are accessible to other users. This solution is discussed later in the book.)
In any case, the method used to correct the typing problem is to first type the line number containing the error to activate that line for editing. In the preceding example, we did so by typing the number 1, shown in bold. Then we used the change command, also shown in bold, observing the proper syntax:

\[ c/old\_value/new\_value \]

After the change is made to the first appearance of old_value in the current line, Oracle redisplays the current line with the change made. Note that the change will be made to the first appearance of old_value only. If the change must be made to a specific place in the line, more characters can be added to the old_value parameter, as appropriate. Finally, the corrected text can be reexecuted by entering a slash (/) at the prompt, as indicated, or by entering the command `run` on the SQL*Plus command line.

**TIP**
If you ever get confused about the difference between the use of the slash and semicolon, remember that the slash command reruns the code currently in your SQL*Plus operating buffer, whereas the semicolon is used to end a SQL statement you type into the buffer.

**Using a Text Editor**
Oracle makes provisions for you to use your favorite text editor to edit the statement created in `afiedt.buf`, the file in which SQL*Plus stores the most recently executed SQL statement. You simply type `edit` (abbreviated `ed`). This action causes Oracle to bring up the SQL statement from `afiedt.buf` into the operating system’s default text editor. On UNIX systems, that text editor is usually VI or EMACS, whereas Windows environments use Notepad. To change the text editor used, issue the `define _editor=`‘yoursecret’ statement on the SQL*Plus prompt.

**TIP**
You can also define your text editor in the SQL*Plus GUI interface using the Tools | Environment menu option.

Using a text editor rather than the line editor native to SQL*Plus offers many benefits. By using a text editor you know well, you can create a familiarity with SQL*Plus that is useful for adapting to the application. Also, it is helpful with large queries to have the entire block of code in front of you and immediately accessible.
Writing SQL Commands in Scripts

You can write entire queries in a text editor first and then load the queries into SQL*Plus if you want to. When you do this, try to remember to save the script with a .sql extension so that SQL*Plus can identify it easily. Two commands are available to load the file into SQL*Plus. The first is \textit{get}. The \textit{get} command opens the text file specified and places the contents in \texttt{afiedt.buf}. Once the script is loaded, you can execute the command using the slash (/) command. Alternatively, you can use the @ or \textit{start} command, which loads SQL statements from the named file into \texttt{afiedt.buf} and executes them in one step. The methods are shown in the following example, with a script called \texttt{select_emp.sql}:

$$$/home/oracle> sqlplus scott/tiger$$

\texttt{SQL*Plus: Release 8.1.7.0.0 - Production on Fri Jul 06 18:53:11 2001}

\texttt{Copyright (c) Oracle Corporation 1979, 2000. All rights reserved.}

\texttt{Connected to Oracle9i Release 9.0.1.0.0}

\texttt{With the partitioning option}

\texttt{JServer Release 9.0.1.0.0 - Production}

\texttt{SQL> GET select_emp}

\texttt{SELECT * FROM emp}

\texttt{SQL> /}

\begin{verbatim}
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
\end{verbatim}

\texttt{SQL> @select_emp}

\texttt{SELECT * FROM emp}

\begin{verbatim}
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
\end{verbatim}
TIP
The “at” (@) sign in front of your SQL script name in the code block above serves a different purpose than the @ in front of the database name we saw in an earlier example when we started SQL*Plus on the command line. Be sure you don’t confuse the two forms of usage.

Notice that the .sql extension was left off the end of the filename in the line with the get command. SQL*Plus assumes that all scripts containing SQL statements will have the .sql extension, so it can be omitted in the get and the @ commands. You can store SQL commands in text files with other extensions, such as .txt and .lst, but if you do, you have to specify the full filename, including the extension, in the get command. Notice also that after the file is brought in using get, it can then be executed using the slash (/) command. Later in that same code block, we use the @ command to read the same file into afiedt.buf. The contents of the buffer are executed in the same step, which eliminates the need for entering the slash (/) command. Again, we omit the .sql extension. Finally, if you don’t specify the path when typing the filename for the get or @ command, Oracle assumes the file is in whatever directory you were in when you started running SQL*Plus.

TIP
When typing SQL statements in a script that you intend to execute in SQL*Plus, do not put a semicolon (;) at the end of these SQL statements. Instead, put a slash (/) character as the first character on the last line in the script. Do this if you encounter problems where Oracle says it encountered an invalid character (the semicolon) in your script.
Other SQL*Plus Commands to Know

The rest of this discussion focuses on identifying other important commands you should know in SQL*Plus, both for your job and for passing the OCP exam. Let's now take a look at explanations for important SQL*Plus commands to know about.

DESCRIBE tablename

This command returns a description of tablename, including all columns in that table, the datatype for each column, and an indication of whether the column permits storage of NULL values. If you experience ORA-00904 errors, this command is used for determining the names of columns in the table you referenced. This command is synonymous with its abbreviation, desc.

Here’s an example:

```
SQL> describe emp
Name Null? Type
----- -------- ------------
EMPNO NOT NULL NUMBER(4)
ENAME VARCHAR2(10)
JOB VARCHAR2(9)
MGR NUMBER(4)
HIREDATE DATE
SAL NUMBER(7,2)
COMM NUMBER(7,2)
DEPTNO NUMBER(2)
```

LIST

This command is used to list the contents of the current SQL*Plus working buffer, organized by line number. SQL*Plus buffers the last SQL command you issued. If you haven’t entered a SQL command yet, the SP2-0223: No lines in SQL buffer error message is displayed. The current line available for editing and other changes is indicated by an asterisk next to the line number. Here’s an example:

```
SQL> select empno, ename
   2  from emp
   3  where empno < 7700;
EMPNO ENAME
------- ----------
7369 SMITH
7499 ALLEN
7521 WARD
7566 JONES
7654 MARTIN
7698 BLAKE
6 rows selected.
SQL> list
```
1  select empno, ename
2  from emp
3  where empno < 7700

**DEL number**  This command deletes line *number* from the SQL*Plus working buffer (not *number* lines!). Each line in the buffer is preceded by a line number, and the last line in the buffer has an asterisk (*) next to the line number. If you want to delete multiple lines, list each line to be removed separated by a space. Here’s an example:

```
SQL> del 3
SQL> list
  1  select empno, ename
  2  from emp
```

**APPEND string**  This command adds *string* specified to the current line. Blank spaces are permitted in the string, and a leading blank space should be included if the current string already has information in it. The current line is indicated with an asterisk (*) in the output of the append command. See the following append command for displaying current line information along with the contents of the SQL*Plus working buffer:

```
SQL> append where empno < 7700
   2  from emp where empno < 7700
```

**CLEAR BUFFER**  This command clears the contents of the SQL*Plus buffer. Here’s an example:

```
SQL> clear buffer
Buffer cleared
```

**INPUT**  When entered at the SQL prompt, this command enables you to add contents to your SQL*Plus operating buffer at the current line. If the buffer was cleared, you start at the first line. If the buffer has something in it, you start at the beginning of a new line at the end of the buffer. Here’s an example:

```
SQL> input
   1  select ename, sal
   2  from emp
   3  where empno < 7600;
ENAME       SAL
----------  ---------
SMITH       800
ALLEN       1600
```
WARD            1250  
JONES           2975  

TIP  
The *append* command is different from the *input* command because *append* allows you to specify the string you want to append, while *input* is specified by itself so that you can enter the string you want added on the next line.

RUN  This command executes the contents of the SQL*Plus buffer. Here’s an example:

```
SQL> run
  1  select ename, sal
  2  from emp
  3* where empno < 7600
ENOAME  SAL
  ----------  ---------
SMITH     800
ALLEN     1600
WARD      1250
JONES     2975
```

number string  When a number is entered in SQL*Plus followed by a string of characters, SQL*Plus adds the string you specify to the operating buffer as the line number you indicated. If the line number already exists, Oracle replaces it. If the line number indicated is not contiguous with the existing lines in the buffer, SQL*Plus adds the string as the last line number in the buffer. Here’s an example:

```
SQL> 6 new line being added
SQL> list
  1  select ename, sal
  2  from emp
  3  where empno < 7600
  4* new line being added
SQL> 2 from jason.emp
SQL> list
  1  select ename, sal
  2  from jason.emp
  3  where empno < 7600
  4* new line being added
```
SPOOL {filename|OFF|OUT}   This command writes all output shown in SQL*Plus following issuance of the spool filename command to a text file identified by filename. If no filename extension is specified, SQL*Plus appends the .lst extension. When the off or out keyword is specified, spooling SQL*Plus output to a file is turned off. Here’s an example:

```sql
SQL> spool jason.out
SQL> select ename, sal
    2   from emp
    3   where empno < 7600;
ENAME   SAL
------- -------
SMITH    800
ALLEN    1600
WARD     1250
JONES    2975
SQL> spool off
SQL> exit
C:\WINDOWS> type jason.out
SQL> select ename, sal
    2   from emp
    3   where empno < 7600;
ENAME   SAL
------- -------
SMITH    800
ALLEN    1600
WARD     1250
JONES    2975
SQL> spool off
```

SAVE filename   This command places the contents of your SQL*Plus buffer into a text file called filename. If no filename extension is specified, SQL*Plus appends .sql.

EXIT   This command exits the SQL*Plus interface and returns to the operating system.

TIP
You can see where having the ability to edit your SQL commands using your favorite text editor is a handy feature of SQL*Plus that makes it possible to avoid learning all the commands of SQL*Plus. Nevertheless, be sure you understand the basics of entering SQL using SQL*Plus before taking the OCP exam.
For Review

1. Be sure you know the two mechanisms available for entering and modifying SQL statements within SQL*Plus.

2. Know how to use the `edit` command in the SQL*Plus command line and how to load and run the contents of SQL scripts into SQL*Plus.

3. Understand how to use the other SQL*Plus commands identified in this section.

Exercises

1. You are modifying a text string on line 3 of your SQL*Plus buffer. Which of the following choices best identifies the method you must use if the `edit` command is used?
   - A. Modify the code block using your favorite text editor.
   - B. First refer to the line number; then use the `change` command.
   - C. First delete the line using the `del` command; then refer to the line number.
   - D. Load the SQL you intend to modify using the `input` command.

2. You would like to list the columns found in an Oracle table. Which of the following SQL*Plus commands are useful for this purpose?
   - A. `get`
   - B. `input`
   - C. `describe`
   - D. `spool`

3. This command displays the contents of your SQL*Plus buffer: ____________________

4. This is the name of the file Oracle stores the contents of your SQL*Plus buffer in: ____________________

Answer Key

1. A. 2. C. 3. list 4. afiedt.buf
Chapter Summary
This chapter ambitiously takes you from an introduction to the Oracle database through some basic techniques used in `select` statements. You learned about the theory behind relational database systems such as Oracle’s and how they differ from earlier systems for data storage and retrieval. The concept of a table was presented, along with common Oracle datatypes used in those tables. The chapter also described the basic architecture of an Oracle database system and covered such factors as what an object-relational RDBMS is and some of the features for developing code in Oracle’s proprietary programming language, PL/SQL. You then focused your attention on the use of `select` statements. We discussed the use of the column and table clauses as well.

Two-Minute Drill
- Data is retrieved from Oracle using `select` statements.
- The syntax for a `select` statement consists of `select ... from ...;`.
- Expressions appearing after the keyword `select` are part of the column clause, and are usually the names of columns from the table storing the data you wish to retrieve.
- Expressions appearing after the `from` keyword are part of the table clause, and are usually the names of tables you want to retrieve data from.
- When you’re entering a `select` statement from the prompt using SQL*Plus, a semicolon (`;`) at the end of the statement or a slash (`/`) at the beginning of the first empty line appearing after the statement in your operating buffer must be used to terminate the statement.
- Arithmetic operations can be used to perform math operations on data selected from a table or on numbers using the DUAL table.
- The DUAL table is a table with one column and one row used to fulfill the syntactic requirements of SQL `select` statements.
- Values in columns for particular rows may be empty (NULL).
- If a column contains a NULL value, you can use the `nvl( )` function to return meaningful information instead of an empty field.
- Aliases can be used in place of the actual column name or to replace the appearance of the function name in the header.
- Output from two columns can be concatenated together using a double pipe (`||`). Alternately, the `concat( )` function can be used for this purpose.
- SQL commands can be entered directly into SQL*Plus on the command line.
- You can edit mistakes in SQL*Plus with the change command. If a mistake is made, the change (c/old/new) command is used.
- Alternatively, the edit (ed) command can be used to make changes in your favorite text editor.
- You can specify your favorite text editor by issuing the define _editor command at the prompt.
- Use the acronym PEMDAS to remember the correct order for operator precedence.
- There are a host of commands available in SQL*Plus that are not part of Structured Query Language to be aware of. A few to pay close attention to include:
  - get for retrieving SQL scripts into SQL*Plus
  - run for executing retrieved SQL scripts
  - @ for getting and running a script in one operation
  - describe for listing the columns in a particular table, along with their datatypes
  - spool for telling SQL*Plus to write the contents of your session to a file
Fill-in-the-Blank Questions

1. This term refers to a logical grouping of tables according to the user who created the tables: ___________________________

2. When you want to perform an operation on two expressions, you can query this table: ____________________________

3. A command-line tool you will use frequently to access Oracle is called: ____________________________

4. The function whose work is performed by placing two pipe characters (||) together is called: ____________________________

5. The Oracle component handling the actual obtainment of data you request is called: ____________________________

6. The command set you request data from Oracle with is called: ____________________________

Chapter Questions

1. You are formulating queries in SQL*Plus. Which of the following statements correctly describes how to specify a column alias?
   A. Place the alias at the beginning of the statement to describe the table.
   B. Place the alias after each column, separated by a space, to describe the column.
   C. Place the alias after each column, separated by a comma, to describe the column.
   D. Place the alias at the end of the statement to describe the table.

2. You wish to use a function in your column clause of a SQL statement. The nvl() function accomplishes which of the following tasks?
   A. Assists in the distribution of output across multiple columns
   B. Enables you to specify alternate output for non-NULL column values
   C. Enables you to specify alternate output for NULL column values
   D. Nullifies the value of the column output
3. Output from a table called PLAYS with two columns, PLAY_NAME and AUTHOR, is shown next. Which of the following SQL statements produced it?

```sql
PLAY_TABLE
-----------------------
"Midsummer Nights Dream", SHAKESPEARE
"Waiting For Godot", BECKETT
"The Glass Menagerie", WILLIAMS

A. select PLAY_NAME || AUTHOR from PLAYS;
B. select PLAY_NAME, AUTHOR from PLAYS;
C. select PLAY_NAME || ', ' || AUTHOR from PLAYS;
D. select PLAY_NAME || ', ' || AUTHOR play_table from PLAYS;
```

4. You are configuring your SQL*Plus working environment. Issuing the define _editor='emacs' will produce which of the following outcomes?

A. The EMACS editor will become the SQL*Plus default text editor.
B. The EMACS editor will start running immediately.
C. The EMACS editor will no longer be used by SQL*Plus as the default text editor.
D. The EMACS editor will be deleted from the system.

5. You are using SQL*Plus to execute some math functions. What is the appropriate table to use when performing arithmetic calculations on values defined within the select statement (not pulled from a table column)?

A. EMP
B. The table containing the column values
C. DUAL
D. An Oracle-defined table
6. You wish to use SQL*Plus to connect to the Oracle database. Which of the following choices does not indicate a component you must specify when logging into Oracle?
   A. The `sqlplus` keyword
   B. The username
   C. The password
   D. The database name

7. Review the following output from a SQL*Plus session:

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYMPTOM</td>
<td>NOT</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>CAUSE</td>
<td>NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>TREATMENT</td>
<td></td>
<td>VARCHAR2(9)</td>
</tr>
</tbody>
</table>

Which of the following keywords likely produced the output above?
   A. `describe`
   B. `get`
   C. `run`
   D. `spool`
Fill-in-the-Blank Answers

1. Schema
2. DUAL
3. SQL*Plus
4. concat( )
5. RDBMS or relational database management system
6. SQL or structured query language

Answers to Chapter Questions

1. B. Place the alias after each column, separated by a space, to describe the column.

Explanation: Aliases do not describe tables; they describe columns, which eliminates choices A and D. Commas are needed between each column appearing in the column clause of the select statement. If a column alias appeared after a column, Oracle would either select the wrong column name, based on information provided in the alias, or return an error.

2. C. Enables you to specify alternate output for NULL column values

Explanation: The nvl( ) function is a simple if-then operation that tests column value output to see whether it is NULL. If it is, nvl( ) substitutes the specified default value for the NULL value. Because this function only operates on one column per call to nvl( ), choice A is incorrect. Choice B is incorrect because it is the logical opposite of choice C. Choice D is incorrect because nvl( ) is designed to substitute actual values for situations where NULL is present, not nullify data.

3. D. select PLAY_NAME||', ' || AUTHOR play_table from PLAYS;

Explanation: This question illustrates the need to read carefully. Because the output specified for the question contains a column alias for the output of the statement, choice D is the only one that is correct, even though choice C also performs the correct calculation. Choice A is incorrect because it specifies an inaccurate concatenation method, and choice B is wrong because it doesn’t specify concatenation at all.
4. A. The emacs editor will become the SQL*Plus default text editor.

Explanation  The `define _editor` statement is designed to define the default text editor in SQL*Plus. Changing the definition will not start or stop the editor specified from running, which eliminates choices B and D. Choice C is the logical opposite of choice A and is therefore incorrect.

5. C. DUAL

Explanation  When all data to be processed by the query is present in the statement, and no data will be pulled from the database, users typically specify the DUAL table to fulfill the syntactic requirements of the `from` clause.

6. D. The database name

Explanation  You needn’t specify the name of the database you wish to connect to. If this information is omitted, then Oracle assumes you want to connect to the local database called ORCL on your machine. All other choices identify a component required for connecting to the Oracle database.

7. A. `describe`

Explanation  The `describe` command produces a listing of all columns in a table, along with their associated datatypes. Choice B is incorrect because `get` merely loads the contents of a script into SQL*Plus memory. Choice C is incorrect because `run` executes a script that has already been loaded into SQL*Plus memory. Finally, choice D is incorrect because the `spool` command is used for writing the commands from a SQL*Plus session issued after the `spool` command to flat file.