CLIENT SERVER COMPUTING WITH ORACLE

SYLLABUS

UNIT I

Basic Concepts, Introduction to Oracle Server - Data Dictionary - Tablespaces and Datafiles - Data Blocks, Extents and Segments - Schema Objects.

UNIT II

SQL SQL*PLUS: Basic SQL

UNIT III

Schema Objects, Data Integrity - Creating and Maintaining Tables - Indexes Sequences Views - Users, Privileges and Roles - Synonyms.

UNIT IV

PL/SQL, PL/SQL - Triggers - Stored Procedures and Functions - Packages - Cursors - Transaction.

UNIT V

Distributed Processing, Distributed Processing - Replication.

UNIT I

LESSON

1

BASIC CONCEPTS

CONTENTS

1.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concepts of oracle
- Discuss how to identify the development needs

- Describe the significance of invoking SQL*PLUS
- Identify and explain the data types
- Discuss the various menus
- Explain the oracle tools and utilities
- Explain the back up and recover in oracle

1.1 INTRODUCTION

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multi-user environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery. The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

1.2 BASIC CONCEPTS OF ORACLE

Every business enterprise maintains large volumes of data for its operations. With more and more people accessing this data for their work the need to maintain its integrity and relevance increases. Normally, with the traditional methods of storing data and information in files, the chances that data loses its integrity and validity are very high.

Oracle is an Object Relational Database Management System (ORDBMS). It offers capabilities of both relational and object-oriented database system. In general, objects can be defined as reusable software codes, which are location independent and perform a specific task on any application environment with little or no change to the code.

1.2.1 Modules of Oracle

The tools/modules provided by Oracle are so user-friendly that a person with minimum skills in the field of computers can access them with ease. The tools are:

- SQL * Plus
- PL/ SQL
- Oracle Forms
- Oracle Report Writer
- Oracle Graphics

SQL*Plus

SQL*Plus is a structured Query Language supported by Oracle. Through SQL*Plus we can store, retrieve, edit, enter and run SQL commands and PL/SQL blocks. Using SQL* Plus we can perform calculations, list column definition for any table and can also format query results in the form of a report.

PL/SQL

PL/SQL is an extension of SQL. PL/SQL block can contain any number of SQL statements integrated with flow of control statements. Thus PL/SQL combines the data manipulating power of SQL with data processing power of procedural languages.

SQL vs. SQL * Plus

SQL is a standard language common to all relational databases. SQL is a database language used for storing and retrieving data from the database. Most Relational Database Management Systems provide extensions to SQL to make it easier for application developers.

SQL*Plus is an Oracle specific program which accepts SQL commands and PL/SQL blocks and executes them. SQL*Plus enables manipulation of SQL commands and PL/SQL blocks. It also performs many additional tasks as well.

Oracle Forms

This tool allows you to create a data entry screen along with suitable menu objects. Thus it is the Oracle Forms tool, which handles data gathering and data validation in a commercial application.

Oracle Report Writer

Report Writer allows programmers to prepare innovative reports using data from the Oracle Structures like tables, views etc. Thus, it is the Report Writer Tool that handles the reporting section of a commercial application.

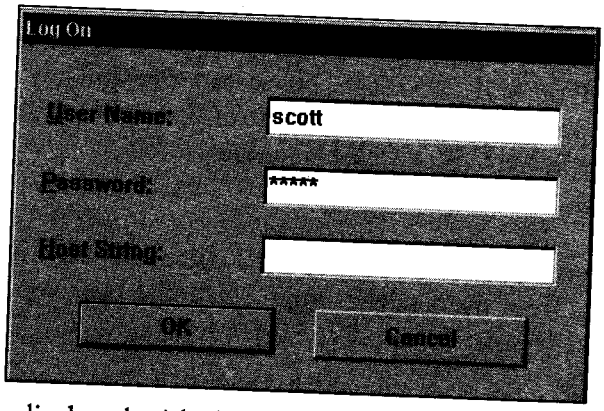
Oracle Graphics

Some of the data can be better represented in the form of graphs. The Oracle Graphics Tool allows programmers to prepare graphs using data from Oracle Structures like tables, views etc. Oracle Graphics can also be considered as a part of the reporting section of a commercial application.

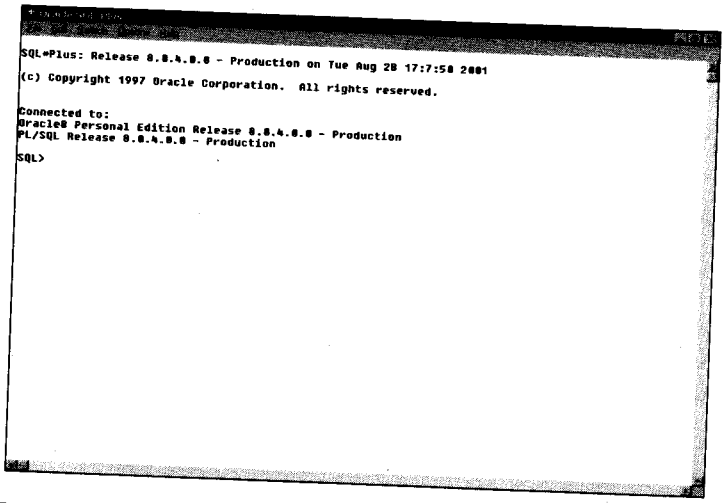
1.3 INVOKING SQL*PLUS

This portion of the lesson shows you SQL * Plus, a tool that allows the handling of a database through individual and interactive execution of the SQL commands. In addition to the direct execution SQL commands, SQL * Plus allows the configuration of the PL/SQL commands. SQL * Plus shows results in the character mode. SQL * Plus is simple to operate and represents the fastest way to query and create quick reports.

To configure SQL * Plus, click on the start button, then on Programs | Oracle Home |Application Development |SQL * Plus. If the user is not connected to the database, the user name, password, and host string are required. For the local database, it is not necessary to give host string. The database is then configured.



Then the SQL * Plus screen is displayed with the SQL> prompt. The prompt is where you enter the SQL commands. The SQL * Plus interface allows a SQL command to occupy several lines. It will execute the command only when the semicolon (;) is typed and the Enter key is pressed. To execute the last command typed in the buffer, the user types slash (/) and press Enter key.



1.4 DATA TYPES

Before continue our discussion of commands, let's turn our attention to the data types that can be stored in an Oracle database. In order to create a table we need to specify a datatype for individual columns in the create table command. When creating a table you must provide certain information, such as the name of the table, names and datatypes for each column. Oracle supports the following datatypes, to achieve the above requirements.

1.4.1 Character Datatypes

The following are the character datatypes supported by Oracle:

Char Datatype: The char datatype is used when a fixed length character string is required. It can store alphanumeric values. The column length of such a datatype can vary between 1- 2000 bytes. The standard size is 1 byte and maximum size is 255 bytes.

If the user enters a value shorter than the specified length then the database blank-pads to the fixed lengths.

In case, if the user enters a value larger than the specified length then the database would return an error.

Varchar2 Datatype: The varchar2 () datatype supports a variable length character string. It also stores alphanumeric values. The size of this datatype ranges from 1 - 4000 bytes. Using varchar2 saves disk space when compared to char. The varchar2 type must be used for the fields of variable size, with a maximum size of 2000 characters, such as the fields of the memo type. While it has variable size, the maximum size it can occupy must be specified.

Long Data Type: This datatype is used to store variable character length. Maximum size is 2 GB. Long datatype has several characters similar to varchar2 datatype. Its length would be restricted based on the memory space available in the computer. The following restriction needs to be fulfilled when a long datatype attribute is cast on a column in a table.

Only one column in a table can have long datatype. This should not contain unique or primary key constraints. The column cannot be indexed. The procedures or store procedures cannot accept long datatype as argument. It also cannot be used in the Where, Order By, and Group By clauses.

Number Data Type: The number datatype can store positive numbers, negative, zeros, fixed point numbers, and floating point numbers with a precision of 38.

column_name number	{p=38, s=0}
column_name number (p)	{fixed point}
column_name number (s)	{floating point}

Where p is the precision, which refers to the total number of digits, it varies between 1 to 38; s is the scale width, which refers to number of digits to the right of the decimal point, which varies between -84 to 127.

Date Datatype: Date datatype is used to store date and time in a table. Oracle store dates in a fixed length of 7 bytes each for the century, year, month, day, hour, minute, and seconds. All of this data is stored for each field of date datatype. Default date datatype is "dd-mon-yy". If only date is given, the hour is written as 12:00 A.M. If only the time is given, the date is stored as the first day of the current month. To view the system's date and time we can use the SQL function called sysdate (). Valid date is from Jan 1, 4712 BC to Dec 31, 4712 AD.

Raw Datatype: Raw datatype is used to store byte-oriented data like binary data or byte strings and the maximum size of this datatype is 2000 bytes. While using this datatype the size should be mentioned because by default it does not specify any size. Only storage and retrieval of data are possible, manipulation of data cannot be done. Raw datatype can be indexed.

Long Raw Datatype: Long raw datatype is used to store binary data of variable length, which can have a maximum size of 2 GB. This datatype cannot be indexed. Further all limitation faced by long datatypes also holds good for long raw datatype.

In addition to the above mentioned Oracle supports:

LOB Datatype: LOB is otherwise known as Large Object datatypes. This can store unstructured information such as sound clips, video files etc., up to 4 GB in size. They allow efficient, random, piece-wise access to the data. The LOB types store values, which are known as locators. These locators

store the location of large objects. LOBs can be either internal or external depending on their location with regards to the database. Data stored in a LOB column is known as LOB value.

The different internal LOBs are mentioned below:

CLOB: A column with its data type as CLOB stores character objects with single byte characters. It *cannot contain character sets of varying widths. A table can have multiple columns with CLOB* as its datatype.

BLOB: A column with its datatype as BLOB can store large binary objects such as graphics, video clips and sound files. A table can have multiple columns with BLOB as its datatype.

BFILE: A BFILE column stores file pointer to LOBs managed by file system external to the database. A BFILE column may contain filenames for photos stored on a CD-ROM.

1.5 MENUS

1.5.1 File Menu

The file menu contains the following options:

Open: Opens a file of SQL commands that was previously written with the extension of .sql.

Save: Allows the writing to the buffer of a txt file that receives the extension .sql as default. The Replace sub-option changes the contents of an existing file with contents of the buffer. The Append sub-option adds the contents of the buffer to the specified file.

SaveAS: Writes the contents of the buffer or the file that was configured with another name.

Spool: Stores the result of a query in a file. As the default, the file created has the extension .lst. To disable Spool for queries, you must activate the Spool Off Option.

Run: Lists and executes the SQL command or a PL/SQL block that is stored in the buffer.

Cancel: Interrupts the operation that is being executed.

Exit: Makes a commit of all the changes made to the database.

1.5.2 Edit Menu

The edit menu includes the following operations:

Copy: Sends the selected text to the Windows clipboard.

Paste: Pastes the contents of the clipboard to the command line of SQL * Plus, with maximum size of 3,625 characters.

Clear: Clears the contents of the screen and the buffer.

Editor: Opens an editor in which the files of PL/SQL commands and their settings can be edited. As the default, Windows Notepad is selected. The name of the file defaults to afiedt.buf. Whenever the editor is called by the Invoke Editor option, the contents of the buffer are automatically transferred to it.

1.5.3 Find Menu

The find menu contains the Find and Find Next options. Find opens a dialog box in which the user can type text to search for. After clicking on the OK button in this dialog box, the occurrence of the text is highlighted; Find Next finds the next occurrence. The search is always initiated from the current screen.

1.5.4 Option Menu

The option menu allows you to change SQL * Plus elements. Environment option opens a Dialog box that has two parts. The define part option list allows you to change the environment elements, such as variables and characteristics of SQL * Plus, the printing of column headers, and the formatting of numeric fields.

The screen buffer area controls the number of characters and lines that are maintained and displayed by SQL * Plus. As the default, the buffer is adjusted to display up to 1,000 lines, with 100 characters. The parameters can also be changed through the SET command directly from the edit line.

1.6 ORACLE TOOLS

1.6.1 Standalone Tools

Various tools are available to address specific environments or specific market requirements.

Development of applications commonly takes place in Java (using Oracle JDeveloper) or through PL/SQL (using, for example, Oracle Forms and Oracle Reports). Oracle Corporation has started a drive toward 'wizard'-driven environments with a view to enabling non-programmers to produce simple data-driven applications.

Oracle SQL Developer, a free graphical tool for database development, allows developers to browse database objects, run SQL statements and SQL scripts, and edit and debug PL/SQL statements. It incorporates standard and customized reporting.

A list of some of the binaries and scripts supplied by Oracle Corporation to operate with/alongside Oracle databases and associated software appears on the Oracle executables web-page.

1.6.2 Administration Tools

The database administrator has several choices for tools to use when managing an Oracle distributed database system:

- Enterprise Manager
- Third-party Administration Tools
- SNMP Support

Enterprise Manager

Enterprise Manager is Oracle's database administration tool that provides a graphical user interface (GUI). Enterprise Manager provides administrative functionality for distributed databases through an easy-to-use interface. You can use Enterprise Manager to:

- Administer multiple databases. You can use Enterprise Manager to administer a single database or to simultaneously administer multiple databases.
- Centralize database administration tasks. You can administer both local and remote databases running on any Oracle platform in any location worldwide. In addition, these Oracle platforms can be connected by any network protocols supported by Oracle Net.
- Dynamically execute SQL, PL/SQL, and Enterprise Manager commands. You can use Enterprise Manager to enter, edit, and execute statements. Enterprise Manager also maintains a history of statements executed.

  Thus, you can reexecute statements without retyping them, a particularly useful feature if you need to execute lengthy statements repeatedly in a distributed database system.
- Manage security features such as global users, global roles, and the enterprise directory service.

Third-party Administration Tools

Currently more than 60 companies produce more than 150 products that help manage Oracle databases and networks, providing a truly open environment.

SNMP Support

Besides its network administration capabilities, Oracle Simple Network Management Protocol (SNMP) support allows an Oracle database server to be located and queried by any SNMP-based network management system. SNMP is the accepted standard underlying many popular network management systems such as:

- HP's OpenView
- Digital's POLYCENTER Manager on NetView
- IBM's NetView/6000
- Novell's NetWare Management System
- SunSoft's SunNet Manager

1.7 ORACLE UTILITIES

Oracle offers the industry's most complete and integrated set of tools for application development, database development, or business intelligence to support any development approach, technology platform, or operating system.

Oracle also provides a variety of free tools to help database and application developers streamline Web application and database development, and make it easy for .NET developers to deploy Oracle-based applications and Web services on the Windows platform.

There are three utilities supplied along with Oracle Server are:

- Export
- Import
- SQL* Loader

They are supplied as part of the Oracle Software. It need not have to be purchased or downloaded separately. They are available as .exe files in the BIN directory and can be executed by typing their name before command prompt. In this lesson, we will be discussing about syntaxes and the usage of these utilities.

1.7.1 Exporting Database Information

This utility can be used to transfer data objects between oracle databases. The objects and the data in Oracle database can be moved to other Oracle database running even on a different hardware and software configurations.

The export utility copies database definitions and actual data into an operating system file (export file). The export file is an Oracle binary-format dump file (with .dmp extension), which is normally created on disk or tape. Before exporting we must ensure that there is enough space available on the disk or tape used.

Exported dump files can be read only by using the Import utility of Oracle. We cannot use earlier versions of import utility for importing the data exported using current version (Versions of Oracle utilities also change along with the Oracle Versions).

EXP command can be used to invoke export utility interactively without any parameters. (Requests the user to enter the value). Otherwise parameters can be specified in a file called parameter file. We can use more than one parameter file at a time with exp command.

Syntax:

exp PARFILE = filename

Parameter file is a simple text file creating using any text editor.

The exports are three types: Full, Owner, and Table.

- ***Full export*** exports all the objects, structures and data within the database for all schemas.
- ***Owner export*** exports only the objects owned by specific user account.
- ***Table export*** exports only tables owned by a specific user account.

To export a table we can run EXP utility either interactively or by putting all the parameters for the export on the command line. In interactive mode just type EXP before the command prompt and answer the questions when prompted, otherwise the parameters can be typed on the command line as shown below.

EXP scott/tiger file=emp.dmp tables=(EMP) log= error.log

In the above example SCOTT/TIGER is the username and password respectively.

emp.dat is the file into which exporting is done. This file is created in the current folder, to create it in a different folder we need to mention the complete path. Ex. C:\sample\dept.dmp.

Tables parameter takes table names as it value, to export more than one table their names need to be separated by a comma. Example tables = (EMP, DEPT, SALGRADE) to export tables EMP, DEPT and SALGRADE.

Log parameter is optional; we give a file name as its value. This file is used to write errors if any occur while exporting a table.

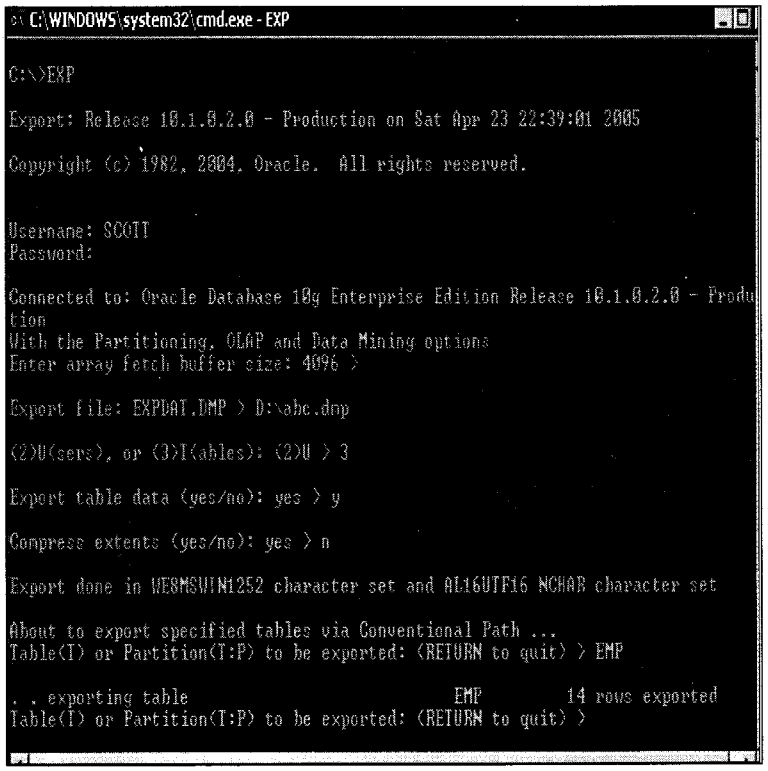


Figure 1.1: Explains how to use EXP Utility in Interactive Mode

Figure 1.2 illustrates exporting data from multiple tables non-interactively (by giving the parameters in the command).

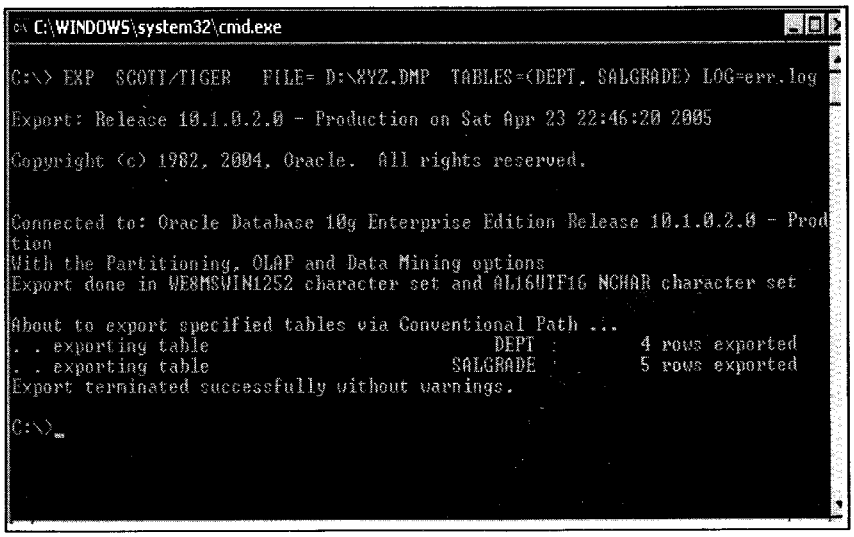


Figure 1.2: Illustrates Exporting Data from Multiple Tables Non-interactively

Exporting data from a table conditionally:

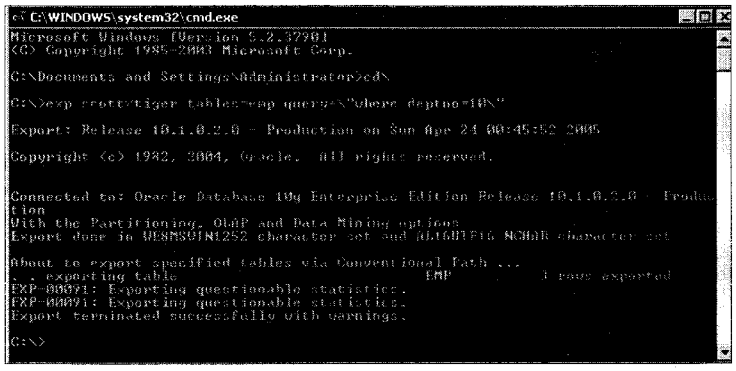


Figure 1.3: Gives the Screen Shot to Export Employees of Deptno = 10

Suppose if you want to export data of employees with salary less than 2000 then the following query can be used.

EXP SCOTT/TIGER FILE= abc.dmp TABLES=emp QUERY=\" WHERE SAL < 2000 \"

1.7.2 Importing Database Information

This utility is used to extract objects (tables etc) from the export file (.dmp file) created using EXP utility.

IMP command can be used to invoke import utility interactively without any parameters. (Which Requests the user to enter the value). Otherwise parameters can be specified in a file called parameter file. We can use more than one parameter file at a time with exp command.

Syntax:

imp username/password PARFILE = filename

(Or)

imp PARFILE = filename

Parameter file is a simple text file creating using any text editor.

Figure 1.4(a) and 1.4(b) explains the usage of IMP utility with parameters (non-interactively) in the command-line.

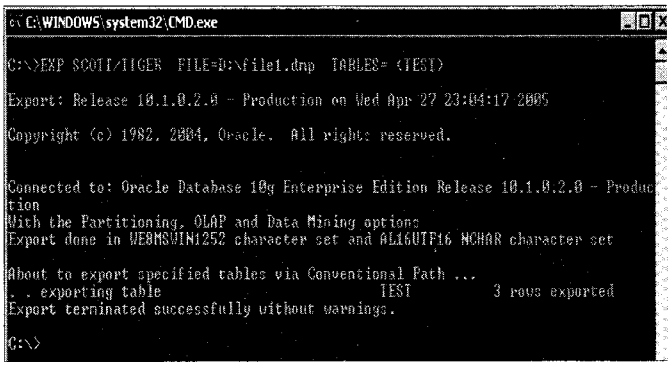


Figure 1.4(a): Explains the Usage of IMP Utility with Parameters

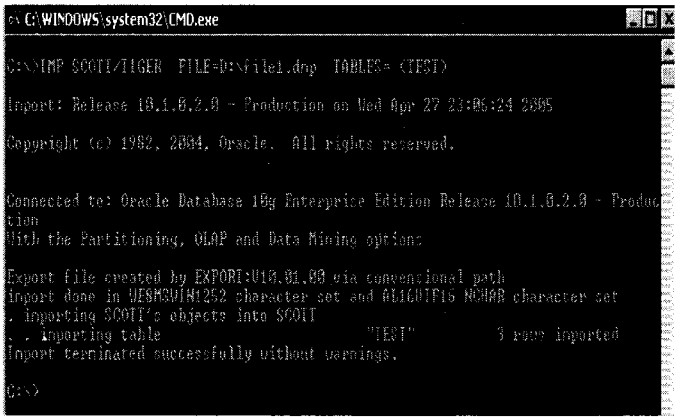


Figure 1.4(b): Explains the Usage of IMP Utility with Parameters

It is possible to import dump created using an earlier version can be imported using the later version utility. We should not use later version utilities to export data from earlier database versions. But an earlier utility can be used to export later versions of database.

1.7.3 Loading Data from Foreign Files

It is an Oracle utility used for moving bulk data from external files into the Oracle database. Data from any text file can be loaded into database. SQL*Loader reads data from an external file and loads data into an existing table while the Oracle database is open.

SQL*Loader Datatypes: SQL*Loader uses the following datatypes.

1. CHAR
2. DECIMAL
3. INTEGER

SQL*Loader require two input files a control file and another data file. The control file is a text file details the task to be carried out by the SQL*Loader. It tells the SQL*Loader where data is available how to parse and interpret it also where to insert it. The data file contains the data to be loaded.

A control file may be vaguely divided into three sections:

1. First section contain INFILE clause in this we specify where input data is located.
2. The second section have INTO TABLE block that details the table and column names into which data is stored.
3. Third section is optional, If present it contain input data.

SQL* Loader assumes that data in data file is organized as records. Based on the record type data files could be categorized into:

Fixed record files: All the records are of same(fixed) length

Variable record files: Records are of varying length and Streamed record files.

Note: If the data is specified in the control file, then we write INFILE * and the data is treated as streamed record format and the records separated by default record terminator.

Examples

The following example explains how to load data from file with fixed record format.

Creating Control File

Using DOS editor or notepad editor you can create data file as shown below:

LOAD DATA

INFILE 'mydata1.dat ' "fix 18"

fields terminated by ','

(sno ,sname , course)

Save this file under the name myctrl1.ctl into the current directory.

Creating Data File

In the DOS editor or Notepad you can create data file as shown below:

1001,RAJAN, ASP,

1002,KISHAN, J2EE,

1003,PRABHU, JSP,

1004,PRANAY, ORACLE,

1005,JOHN, APPS,

1006,MARTIN, ORACLE

Save the above content as file with the name "mydata1.dat"

The following figure explains how data can be loaded into a table. The data loaded here is of fixed-length records

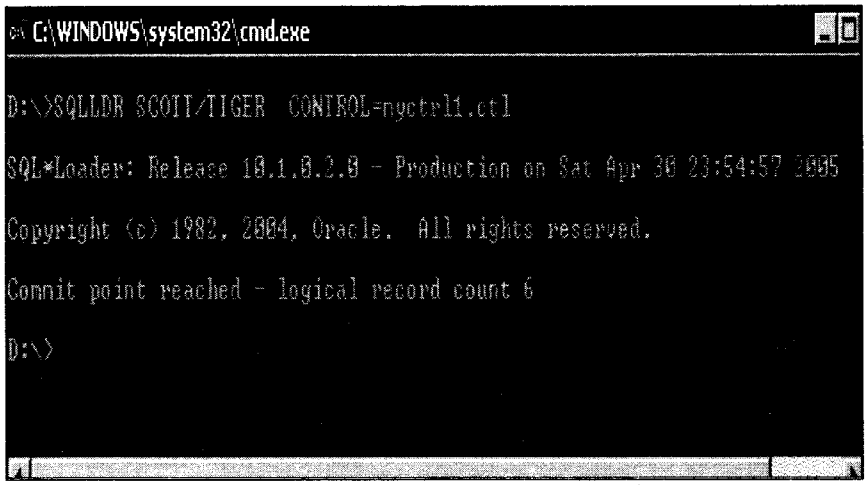


Figure 1.5: Explains how Data can be Loaded into a Table

To cross check whether loading was proper we execute simple SELECT statement as given below.

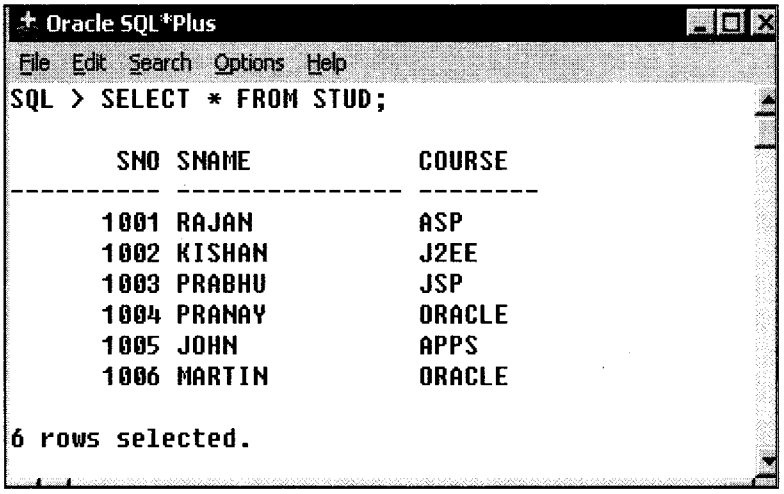


Figure 1.6: The Screen to Check Whether Loading was Proper

1.8 BACKUP AND RECOVER

A backup is a copy of data. This copy can include important parts of the database such as the control file and datafiles. A backup is a safeguard against unexpected data loss and application errors. If you lose the original data, then you can reconstruct it by using a backup.

Backups are divided into physical backups and logical backups. Physical backups, which are the primary concern in a backup and recovery strategy, are copies of physical database files. You can make physical backups with either the Recovery Manager (RMAN) utility or operating system utilities.

In contrast, logical backups contain logical data (for example, tables and stored procedures) extracted with the Oracle Export utility and stored in a binary file. You can use logical backups to supplement physical backups.

To restore a physical backup of a datafile or control file is to reconstruct it and make it available to the Oracle database server. To recover a restored datafile is to update it by applying archived redo logs and online redo logs, that is, records of changes made to the database after the backup was taken. If you use RMAN, then you can also recover restored datafiles with incremental backups, which are backups of a datafile that contain only blocks that changed after a previous incremental backup.

After the necessary files are restored, media recovery must be initiated by the user. Media recovery can use both archived redo logs and online redo logs to recover the datafiles. If you use SQL*Plus, then you can run the RECOVER command to perform recovery. If you use RMAN, then you run the RMAN RECOVER command to perform recovery.

Unlike media recovery, Oracle performs crash recovery and instance recovery automatically after an instance failure. Crash and instance recovery recover a database to its transaction-consistent state just before instance failure.

By definition, crash recovery is the recovery of a database in a single-instance configuration or an Oracle Real Application Clusters configuration in which all instances have crashed. In contrast,

instance recovery is the recovery of one failed instance by a live instance in an Oracle Real Application Clusters configuration.

Crash and instance recovery involve two distinct operations: rolling forward the current, online datafiles by applying both committed and uncommitted transactions contained in online redo records, and then rolling back changes made in uncommitted transactions to their original state. Because crash and instance recovery are automatic, this manual will not discuss these operations.

Figure 1.7 illustrates the basic principle of backing up, restoring, and performing media recovery on a database.

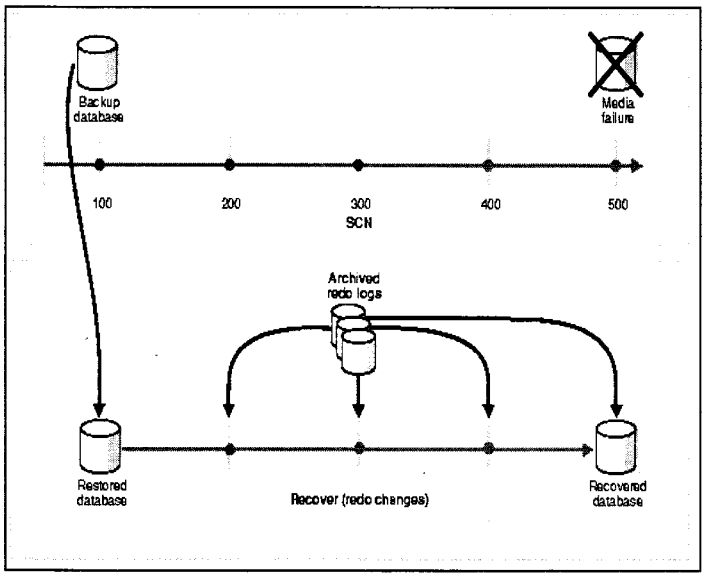


Figure 1.7: Restoring and Recovering a Database

Check Your Progress

Fill in the blanks:

1. Oracle products are based on a concept known as the .....
2. SQL*Plus is a structured Query Language supported by .....
3. The char datatype is used when a ..... length character string is required.
4. The option menu allows you to ..... SQL * Plus elements.
5. Enterprise Manager is Oracle's database administration tool that provides a .....
6. Oracle provides a variety of free tools to help ..... and application developers streamline.

1.9 LET US SUM UP

Oracle is an Object Relational Database Management System (ORDBMS). It offers capabilities of both relational and object-oriented database system. In general, objects can be defined as reusable software codes, which are location independent and perform a specific task on any application environment

with little or no change to the code. In addition to the direct execution SQL commands, SQL * Plus allows the configuration of the PL/SQL commands. SQL * Plus shows results in the character mode. Before continue our discussion of commands, let's turn our attention to the data types that can be stored in an Oracle database.Oracle offers the industry's most complete and integrated set of tools for application development, database development, or business intelligence to support any development approach, technology platform, or operating system. The export utility can be used to transfer data objects between oracle databases running even on a different hardware and software configurations. Exported dump files can be read only by using the Import utility of Oracle. Import utility is used to extract objects (tables etc) from the export file (.dmp file) created using EXP utility. SQL*Loader is an Oracle utility used for moving bulk data from external files into the Oracle database. SQL*Loader require two input files a control file and another data file. The control file is a text file details the task to be carried out by the SQL*Loader. It tells the SQL*Loader where data is available how to parse and interpret it also where to insert it. The data file contains the data to be loaded. A backup is a safeguard against unexpected data loss and application errors. Backups are divided into physical backups and logical backups. Physical backups, which are the primary concern in a backup and recovery strategy, are copies of physical database files. logical backups contain logical data (for example, tables and stored procedures) extracted with the Oracle Export utility and stored in a binary file. To restore a physical backup of a datafile or control file is to reconstruct it and make it available to the Oracle database server. To recover a restored datafile is to update it by applying archived redo logs and online redo logs, that is, records of changes made to the database after the backup was taken. There are media, crash and instance recovery.

1.10 KEYWORDS

Export Utility: It is a utility which can be used to transfer data objects between oracle databases running even on a different hardware and software configurations.

Import Utility: It is a utility which is used to extract objects (tables etc) from the export file (.dmp file) created using EXP utility.

SQL*Loader: It is an Oracle utility used for moving bulk data from external files into the Oracle database.

Control File: It is a text file details the task to be carried out by the SQL*Loader.

Data File: It is a file which contains the data to be loaded.

Backup: It is a safeguard against unexpected data loss and application errors.

Logical Backup: It contains logical data (for example, tables and stored procedures) extracted with the Oracle Export utility and stored in a binary file.

Physical Backup: It is the process in which the primary concern is to copy the physical database files.

1.11 QUESTIONS FOR DISCUSSION

1. What are the basic modules in oracle?
2. Explain the processor to invoke SQL*PLUS.
3. What are basic data types in oracle?

4. Discuss the file menu and edit menu in SQL*PLUS.
5. What is import/export and why does one need it?
6. How does one use the import/export utilities?
7. Can one import/export between different versions of Oracle?

Check Your Progress: Model Answers

1. 'Client/Server Technology'
2. Oracle
3. Fixed
4. Change
5. graphical user interface
6. database

1.12 SUGGESTED READINGS

Dave Moore, *Oracle Utilities,* Rampant TechPress

Kent Crotty and Donald K. Burleson, *Oracle Best Practices: Practical Standards for Success,* Rampant Techpress

Fred D. Rolland , *Relational Database Management with Oracle,* Addison-Wesley

Robert G. Freeman and Steve Karam, *Easy Oracle Jumpstart: Oracle Database Management Concepts and Administration,* Rampant Techpress

Bill Pribyl, *Learning Oracle PL/SQL,* O'Reilly Media

Steven Feuerstein , *Oracle PL/SQL Programming,* O'Reilly Media

LESSON

2

INTRODUCTION TO ORACLE SERVER

CONTENTS

2.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of data dictionary
- Discuss how to identify the tablespaces and data files
- Describe the data blocks, extents and segments
- Identify and explain the schema objects

2.1 INTRODUCTION

Oracle products are based on a concept known as the 'Client/Server Technology'. This concept involves segregating the processing of an application between two systems. One performs all activities related to the database (server) and the other performs activities that help user to interact with the application (client).

A client or front-end database application also interacts with the database by requesting and receiving information from the 'database severs'. It acts as an interface between the user and the database. The commonly used front tool of ORACLE is SQL * Plus.

The database server or back end is used to manage the database tables optimally among multiple clients who concurrently request the server for the same data. It also enforces data integrity across all the clients' applications and controls databases access and other security requirements.

Oracle uses the Internet File System, which is Java based application, which enables database to become an Internet development platform. Oracle also provides complete support for building Java applications by offering new versions of the Jdeveloper. The data store in database can be used to build HTML web pages. Multimedia data store in a network-accessible database can be manipulated or modified using the Oracle inter Media Audio, Image and Video Java Client developed Applications.

2.2 DATA DICTIONARY

One of the most important parts of an Oracle database is its **data dictionary**, which is a **read-only** set of tables that provides information about the database. A data dictionary contains:

- The definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on)
- How much space has been allocated for, and is currently used by, the schema objects
- Default values for columns
- Integrity constraint information
- The names of Oracle users
- Privileges and roles each user has been granted
- Auditing information, such as who has accessed or updated various schema objects
- Other general database information

The data dictionary is structured in tables and views, just like other database data. All the data dictionary tables and views for a given database are stored in that database's `SYSTEM` tablespace.

Not only is the data dictionary central to every Oracle database, it is an important tool for all users, from end users to application designers and database administrators. Use SQL statements to access the data dictionary. Because the data dictionary is read-only, you can issue only queries (`SELECT` statements) against it's tables and views.

2.2.1 Structure of the Data Dictionary

The data dictionary consists of the following:

Base Tables

The underlying tables that store information about the associated database. Only Oracle should write to and read these tables. Users rarely access them directly because they are normalized, and most of the data is stored in a cryptic format.

User-Accessible Views

The views that summarize and display the information stored in the base tables of the data dictionary. These views decode the base table data into useful information, such as user or table names, using joins and WHERE clauses to simplify the information. Most users are given access to the views rather than the base tables.

2.2.2 How the Data Dictionary is used

The data dictionary has three primary uses:

- Oracle accesses the data dictionary to find information about users, schema objects, and storage structures.
- Oracle modifies the data dictionary every time that a data definition language (DDL) statement is issued.
- Any Oracle user can use the data dictionary as a read-only reference for information about the database.

2.2.3 How Oracle uses the Data Dictionary

Data in the base tables of the data dictionary *is necessary for Oracle to function.* Therefore, only Oracle should write or change data dictionary information. Oracle provides scripts to modify the data dictionary tables when a database is upgraded or downgraded.

During database operation, Oracle reads the data dictionary to ascertain that schema objects exist and that users have proper access to them. Oracle also updates the data dictionary continuously to reflect changes in database structures, auditing, grants, and data.

For example, if user Kathy creates a table named `parts`, then new rows are added to the data dictionary that reflect the new table, columns, segment, extents, and the privileges that Kathy has on the table. This new information is then visible the next time the dictionary views are queried.

Public Synonyms for Data Dictionary Views

Oracle creates public synonyms for many data dictionary views so users can access them conveniently. The security administrator can also create additional public synonyms for schema objects that are used systemwide. Users should avoid naming their own schema objects with the same names as those used for public synonyms.

Cache the Data Dictionary for Fast Access

Much of the data dictionary information is kept in the SGA in the **dictionary cache**, because Oracle constantly accesses the data dictionary during database operation to validate user access and to verify the state of schema objects. All information is stored in memory using the least recently used (LRU) algorithm.

Parsing information is typically kept in the caches. The COMMENTS columns describing the tables and their columns are not cached unless they are accessed frequently.

Other Programs and the Data Dictionary

Other Oracle products can reference existing views and create additional data dictionary tables or views of their own. Application developers who write programs that refer to the data dictionary should refer to the public synonyms rather than the underlying tables: the synonyms are less likely to change between software releases.

2.2.4 How to use the Data Dictionary

The views of the data dictionary serve as a reference for all database users. Access the data dictionary views with SQL statements. Some views are accessible to all Oracle users, and others are intended for database administrators only.

The data dictionary is always available when the database is open. It resides in the SYSTEM tablespace, which is always online.

The data dictionary consists of sets of views. In many cases, a set consists of three views containing similar information and distinguished from each other by their prefixes:

Table 2.1: Data Dictionary View Prefixes

Prefix	Scope
USER	User's view (what is in the user's schema)
ALL	Expanded user's view (what the user can access)
DBA	Database administrator's view (what is in all users' schemas)

The set of columns is identical across views, with these exceptions:

- Views with the prefix USER usually exclude the column OWNER. This column is implied in the USER views to be the user issuing the query.
- Some DBA views have additional columns containing information useful to the administrator.

2.3 ORGANIZATION OF DATA IN ORACLE

The relational model has three major aspects:

Structures

Structures are well-defined objects that store the data of a database. Structures and the data contained within them can be manipulated by operations.

Operations

Operations are clearly defined actions that allow users to manipulate the data and structures of a database. The operations on a database must adhere to a pre-defined set of integrity rules.

Integrity Rule

Integrity rules are the laws that govern which operations are allowed on the data and structures of a database. Integrity rules protect the data and the structures of a database.

An ORACLE database has both a physical and a logical structure. By separating physical and logical database structure, the physical storage of data can be managed without affecting the access to logical storage structures.

Logical Database Structure

An ORACLE database's logical structure is determined by:

- One or more tablespaces.
- The database's schema objects (e.g., tables, views, indexes, clusters, sequences, stored procedures).

The logical storage structures, including tablespaces, segments, and extents, dictate how the physical space of a database is used. The schema objects and the relationships among them form the relational design of a database.

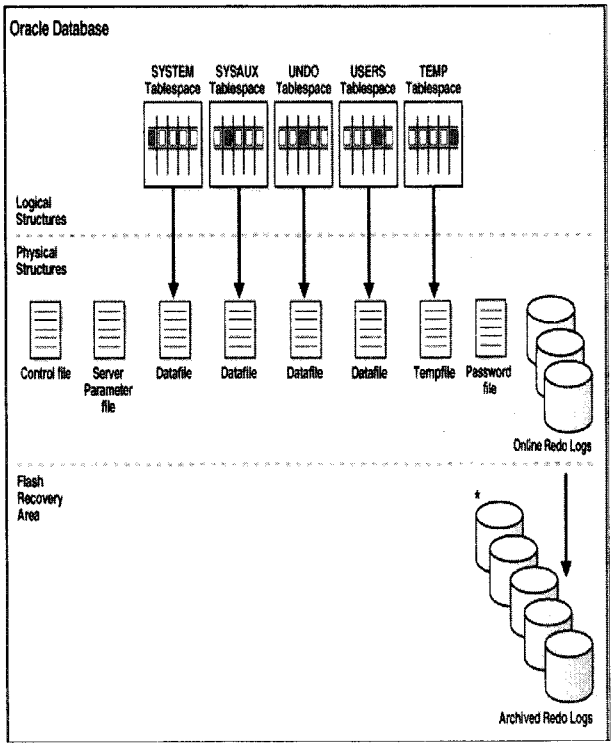


Figure 2.1: Oracle Database Storage Structures

2.3.1 Tablespaces and Data Files

Tablespaces are the primary logical storage structures of any ORACLE database. The usable data of an ORACLE database is logically stored in the tablespaces and physically stored in the data files associated with the corresponding tablespace. Figure 2.2 illustrates this relationship.

Although databases, tablespaces, data files, and segments are closely related, they have important differences:

Databases and Tablespaces

An ORACLE database is comprised of one or more logical storage units called tablespaces. The database's data is collectively stored in the database's tablespaces.

Tablespaces and Data Files

Each tablespace in an ORACLE database is comprised of one or more operating system files called data files. A tablespace's data files physically store the associated database data on disk.

Databases and Data Files

A database's data is collectively stored in the data files that constitute each tablespace of the database. For example, the simplest ORACLE database would have one tablespace, with one data file. A more complicated database might have three tablespaces, each comprised of two data files (for a total of six data files).

2.3.2 Schema Objects

When a schema object such as a table or index is created, its segment is created within a designated tablespace in the database.

For example, suppose a table is created in a specific tablespace using the CREATE TABLE command with the TABLESPACE option. The space for this table's data segment is allocated in one or more of the data files that constitute the specified tablespace. An object's segment allocates space in only one tablespace of a database.

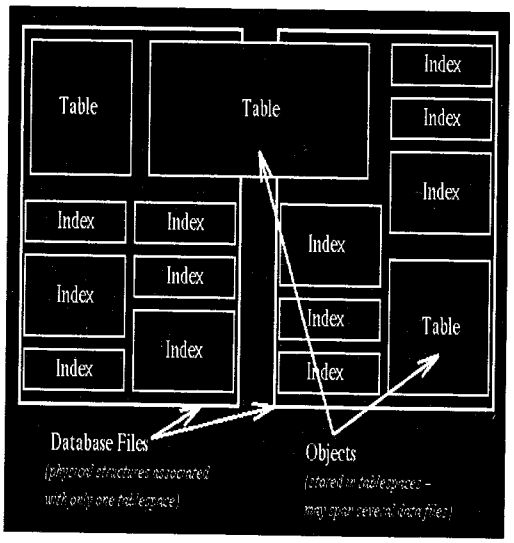


Figure 2.2: Data Files and Tablespaces

A database is divided into one or more logical storage units called tablespaces. A database administrator can use tablespaces to do the following:

- Control disk space allocation for database data.
- Assign specific space quotas for database users.
- Control availability of data by taking individual tablespaces online or offline.
- Perform partial database backup or recovery operations.
- Allocate data storage across devices to improve performance.

Every ORACLE database contains a tablespace named SYSTEM, which is automatically created when the database is created. The SYSTEM tablespace always contains the data dictionary tables for the entire database. You can query these data dictionary tables to obtain pertinent information about the database; for example, the names of the tables that are owned by you or ones to which you have access.

Data files associated with a tablespace store all the database data in that tablespace. One or more datafiles form a logical unit of database storage called a tablespace. A data file can be associated with only one tablespace, and only one database.

After a data file is initially created, the allocated disk space does not contain any data; however, the space is reserved to hold only the data for future segments of the associated tablespace - it cannot store any other program's data. As a segment (such as the data segment for a table) is created and grows in a tablespace, ORACLE uses the free space in the associated data files to allocate extents for the segment.

The data in the segments of objects (data segments, index segments, rollback segments, and so on) in a tablespace are physically stored in one or more of the data files that constitute the tablespace.

Note that a schema object does not correspond to a specific data file; rather, a data file is a repository for the data of any object within a specific tablespace. The extents of a single segment can be allocated in one or more data files of a tablespace (see Figure 2.3); therefore, an object can "span" one or more data files. The database administrator and end-users cannot control which data file stores an object.

2.3.3 Data Blocks, Extents and Segments

ORACLE allocates database space for all data in a database. The units of logical database allocations are data blocks, extents, and segments. Figure 2.4 illustrates the relationships between these data structures.

Data Blocks

At the finest level of granularity, an ORACLE database's data is stored in data blocks (also called logical blocks, ORACLE blocks, or pages). An ORACLE database uses and allocates free database space in ORACLE data blocks. Figure 2.4 illustrates a typical ORACLE data block.

Extents

The next level of logical database space is called an extent. An extent is a specific number of contiguous data blocks that are allocated for storing a specific type of information.

Segments

The level of logical database storage above an extent is called a segment. A segment is a set of extents which have been allocated for a specific type of data structure, and all are stored in the same tablespace. For example, each table's data is stored in its own data segment, while each index's data is stored in its own index segment. ORACLE allocates space for segments in extents. Therefore, when the existing extents of a segment are full, ORACLE allocates another extent for that segment. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk, and may or may not span files. An extent cannot span files, though.

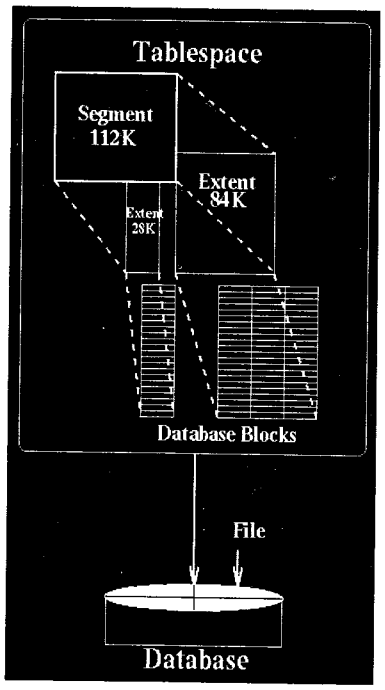


Figure 2.3: The Relationship among Segments, Extents and Data Blocks

ORACLE manages the storage space in the data files of a database in units called data blocks. A data block is the smallest unit of I/O used by a database. A data block corresponds to a block of physical bytes on disk, equal to the ORACLE data block size (specifically set when the database is created - 2048). This block size can differ from the standard I/O block size of the operating system that executes ORACLE.

The ORACLE block format is similar regardless of whether the data block contains table, index, or clustered data. Figure 2.4 shows the format of a data block.

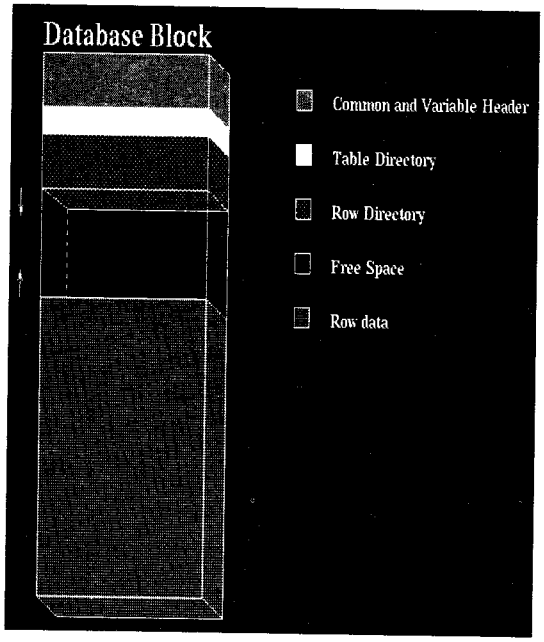


Figure 2.4: Data Block Format

Header (Common and Variable)

The header contains general block information, such as block address, segment type, such as data, index, or rollback. While some block overhead is fixed in size (about 107 bytes), the total block overhead size is variable.

Table Directory

The table directory portion of the block contains information about the tables having rows in this block.

Row Directory

This portion of the block contains row information about the actual rows in the block (including addresses for each row piece in the row data area). Once the space has been allocated in the row directory of a block's header, this space is not reclaimed when the row is deleted.

Row Data

This portion of the block contains table or index data. Rows can span blocks.

Free Space

Free space is used to insert new rows and for updates to rows that require additional space (e.g., when a trailing null is updated to a non-null value). Whether issued insertions actually occur in a given data block is a function of the value for the space management parameter PCTFREE and the amount of current free space in that data block.

Space used for Transaction Entries

Data blocks allocated for the data segment of a table, cluster, or the index segment of an index can also use free space for transaction entries.

Two space management parameters, PCTFREE and PCTUSED, allow a developer to control the use of free space for inserts of and updates to the rows in data blocks. Both of these parameters can only be specified when creating or altering tables and clusters (data segments). In addition, the storage parameter PCTFREE can also be specified when creating or altering indices (index segments).

The PCTFREE parameter is used to set the percentage of a block to be reserved (kept free) for possible updates to rows that already are contained in that block. For example, assume that you specify the following parameter within a CREATE TABLE statement:

- Pctfree 20

  This states that 20\% of each data block used for this table's data segment will be kept free and available for possible updates to the existing rows already within each block.

  After a data block becomes full, as determined by PCTFREE, the block is not considered for the insertion of new rows until the percentage of the block being used falls below the parameter PCTUSED. Before this value is achieved, the free space of the data block can only be used for updates to rows already contained in the data block. For example, assume that you specify the following parameter within a CREATE TABLE statement:

- Pctused 40

  In this case, a data block used for this table's data segment is not considered for the insertion of any new rows until the amount of used space in the blocks falls to 39\% or less (assuming that the block's used space has previously reached PCTFREE).

No matter what type, each segment in a database is created with at least one extent to hold its data. This extent is called the segment's initial extent.

If the data blocks of a segment's initial extent become full and more space is required to hold new data, ORACLE automatically allocates an incremental extent for that segment. An incremental extent is a subsequent extent of the same or incremented size of the previous extent in that segment.

Every non-clustered table in an ORACLE database has a single data segment to hold all of its data. The data segment for a table is indirectly created via the CREATE TABLE/SNAPSHOT command.

Storage parameters for a table, snapshot, or cluster control the way that a data segment's extents are allocated. Setting these storage parameters directly via the CREATE TABLE/SNAPSHOT/ CLUSTER or ALTER TABLE/SNAPSHOT/CLUSTER commands affects the efficiency of data retrieval and storage for that data segment.

2.3.4 Physical Database Structure

An ORACLE database's physical structure is determined by the operating system files that constitute the database. Each ORACLE database is comprised of these types of files: one or more data files, two or more redo log files, and one or more control files. The files of a database provide the actual physical storage for database information.

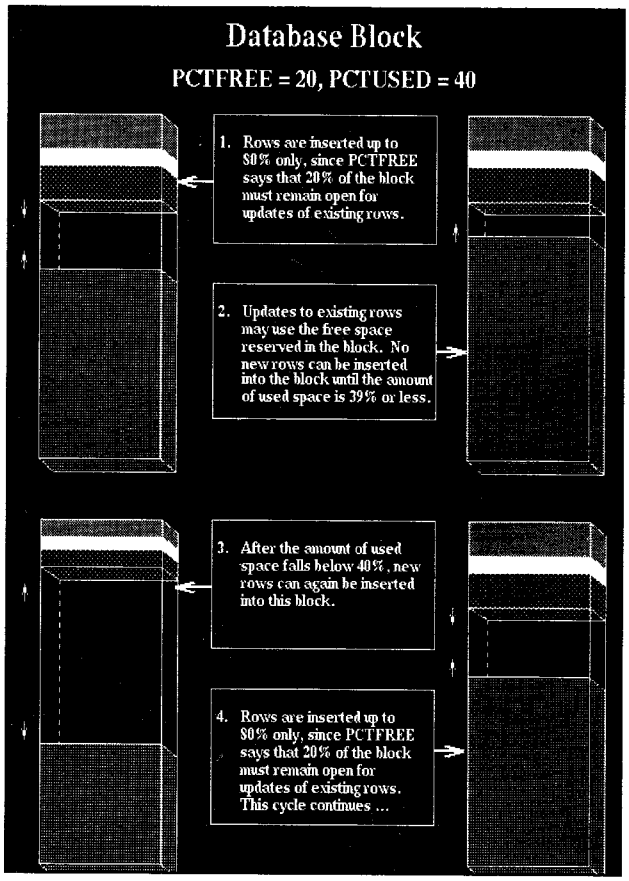


Figure 2.5: Maintaining the Free Space of Data Blocks with PCTFREE and PCTUSED

Check Your Progress

Fill in the blanks:

1. Oracle uses the Internet File System, which is Java based application, which enables database to become an Internet development platform.
2. The data dictionary is structured in tables and views, just like other database data.
3. The views that summarize and display the information stored in the base tables of the data dictionary.
4. An ORACLE database is comprised of one or more logical storage units called tablespaces.

2.4 LET US SUM UP

An Oracle database is a collection of data treated as a unit. A database server is the key to solve the problems of information management. A database server also prevents unauthorized access and provides efficient solutions for failure recovery. The database has logical structures and physical structures. Oracle stores records relating to each other in a table. A table consists of a number of records. Each field occupies one column and each record occupies one row. Related tables are grouped together to form a database. Every table in Oracle has a field or a combination of fields that uniquely identifies each record in the table. This unique identifier is called the primary key, or simply the key. A foreign key is a field or a group of fields in one table whose values match those of the primary key of another table. The process of normalizing data breaks the data down into smaller and smaller tables to reduce redundancy and make retrieving and managing that data more efficient.

2.5 KEYWORDS

Data Dictionary: The data dictionary is structured in tables and views, just like other database data.

Tablespaces: An ORACLE database is comprised of one or more logical storage units called tablespaces

Data Files: Each tablespace in an ORACLE database is comprised of one or more operating system files called data files.

2.6 QUESTIONS FOR DISCUSSION

1. What is the importance of data dictionary in oracle?
2. Explain the relationship among Database, Tablespace and Data file.
3. Explain the terms:
   (a) Data blocks
   (b) Extents
   (c) Segments
4. Discuss the use of schema object.

2.7 SUGGESTED READINGS

David Kreines and Brian Laskey, *Oracle Database Administration: The Essential Reference,* O'Reilly Media

Kent Crotty and Donald K. Burleson, *Oracle Best Practices: Practical Standards for Success,* Rampant Techpress

Fred D. Rolland , *Relational Database Management with Oracle,* Addison-Wesley

Robert G. Freeman and Steve Karam, *Easy Oracle Jumpstart: Oracle Database Management Concepts and Administration,* Rampant Techpress

Steven Feuerstein, Bill Pribyl and Chip Dawes, *Oracle PL/SQL Language,* O'Reilly Media

Bill Pribyl, Learning Oracle PL/SQL, O'Reilly Media

Steven Feuerstein , *Oracle PL/SQL Programming,* O'Reilly Media

UNIT II

LESSON

3

SQL

CONTENTS

3.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of basic SQL
- Discuss the SQL language
- Describe the DDL and DML commands
- Identify and explain the retrieving of data
- Discuss the data definition language

3.1 INTRODUCTION

A database would be of a very little use if users could not interact with it. Therefore, every DBMS must provide some mechanism, perhaps a language and a set of software tools to allow users to submit a request; an engine to process the request; and a mechanism to present the result back to the users.

In early days of DBMS technology, different vendors developed their own customized mechanisms and tools for this purpose. However, over the years, the interaction languages have been standardized. This chapter introduces perhaps the most popular database interaction language called - Structured Query Language - or SQL.

3.2 BASIC SQL

SQL - Structured Query Language - is the basic tool for accessing data in a database. Mastering SQL is the first and most important step you have to take to become a database expert. In recent years, the SQL has left the mainframe and been extended to the desktop. Internet has also popularized this language. Because of its appropriate structure for the client/server architecture, more and more applications and pages that access relational databases are being created with the SQL language.

3.2.1 Oracle and SQL

SQL Language Basics

You will be able to perform most activities related to querying and manipulating a database by learning just a few commands and functions.

The main commands and functions that will be discussed in this part are listed below:

Commands	Functions
SELECT	SUM ()
INSERT	AVG ()
DELETE	MAX ()
UPDATE	MIN ()
COMMIT	COUNT ()
ROLLBACK	SYSDATE ()

Types of SQL Declaration

The SQL declarations, or commands, are divided into two major categories, according to their functionality. There are the Data Definition Language (DDL) commands and Data Manipulation Language (DML) commands.

Data Definition Language (DDL)

The DDL, or Data Definition Language, is a part of the SQL language used to define data and objects in a database. When these commands are used, the Oracle data dictionary receives some entries. The Data Definition Language is used to create an object (e.g., table), alter the structure of an object and also to drop the object created. The concepts relating to Data Definition Language are explained in the following paragraphs.

Table Definition

A table is a unit of storage that holds data in the form of rows and columns. The Data Definition Language used for table definition can be classified into the following four:

- Create table command
- Alter table command
- Truncate table command
- Drop table command

3.3 SQL LANGUAGE

Often pronounced as "sequel", SQL is by far the most popular relational database query language. Almost all the standard relational database management systems support SQL or some of its variants.

SQL is the set of commands that programs and users may use to access data within the database that supports it. Application programs and database tools often allow users to access the database without directly using SQL, however, these applications also translate the actions into SQL under the hood.

SQL has its root in the paper, "A Relational Model of Data for Large Shared Data Banks," published by Dr E. F. Codd in June 1970 in the Association of Computer Machinery (ACM) journal, Communications of the ACM. Incidentally, the paper also contains Codd's model, which is now accepted as the definitive model for relational Database Management Systems.

In 1979, Relational Software, Inc., reincarnated as present day Oracle Corporation, introduced the first commercially available implementation of SQL. Today, SQL is accepted as the standard RDBMS language. The latest SQL standard by ANSI and ISO is often called SQL-92 and sometimes SQL2.

3.3.1 Benefits of SQL

SQL has now become de-facto query language in the relational database world. It has found widespread acceptance by vendors as well as end users. The strengths of SQL benefit the entire range of users including application programmers, database administrators, managers and end users. Some of the benefits of SQL worth noting are described below.

In order that the SQL commands get executed to produce the desired result an SQL interpreter or user-interface program is required. Almost every DBMS provides at least one such user-interface. For example, ORACLE provides TSQL utility program that accepts your SQL commands and gets the same executed on your behalf.

Non-procedural

SQL is a non-procedural language in that the users do not have to specify how the SQL commands are to be executed. SQL commands work like macros that have already been written for the users.

To illustrate the point, consider an example of listing all the records from a table - Student - in which Marks column has value more than 60. A procedural language would works as follows:

1. Start
2. Open the table - Student

3. While End-of-file has not been reached
   - ❖ If marks > 60 then print record
   - ❖ Move to next record
4. Close table
5. End

A non-procedural language is free from writing algorithms like one shown above. A single command can do the job. Here is the SQL equivalent of the above listed algorithm.

```
SELECT * FROM STUDENT WHERE MARKS > 60;
```

Evidently, SQL processes sets of records in one go rather than just one at a time. Moreover, it provides automatic navigation to the data set.

SQL provides easy-to-learn commands that are both consistent and applicable to all users. The basic SQL commands can be learned in a few hours and even the most advanced commands can be mastered in a few days.

Unified Languages

SQL provides commands for a variety of tasks including:

- Creating relational database objects like tables, views and indexes
- Modifying and deleting relational database objects
- Querying database
- Inserting, updating and deleting rows in a table
- Creating, replacing, altering and dropping objects
- Controlling access to the database and its object
- Guaranteeing database consistency and language.

SQL unifies all the above tasks in one consistent language.

Common Language for All Relational Databases

Because all major relational database management systems support SQL, you can transfer all skills you have gained with SQL from one database to another. In addition, since all programs written in SQL are portable, they can often be moved from one database to another with very little modification.

Embedded SQL

Embedded SQL refers to the use of standard SQL commands embedded within a procedural programming language. Embedded SQL is a collection of these commands.

The RDBMSs also provide pre-compilers that support embedded SQL. The SQL pre-compilers interpret embedded SQL statements and translate them into statements that can be understood by procedural language compilers. Some of the SQL pre-compilers (for example available with Oracle) that translate embedded SQL programs into a different procedural language are:

- Pro*Ada precompiler
- Pro*C/C++ precompiler

- Pro*COBOL precompiler
- Pro*FORTAN precompiler
- Pro*Pascal precompiler
- Pro*PL/l precompiler

3.3.2 Database Objects

RDBMS supports two types of data objects.

Schema Objects: A schema is a collection of logical structures of data, of schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema. Schema objects can be created and manipulated with SQL and include the following types of objects.

Cluster	database links	database triggers
Indexes	Packaged	sequences
Snapshots	snapshot logs	stored functions
stored procedures	synonyms	tables
Views		

Non-schema Objects: Other types of objects are also stored in the database and can be created and manipulated with SQL, but are not contained in a schema.

Profiles	roles
rollback segments	table spaces
Users	

3.3.3 Object Naming Conventions

The following rules apply when naming objects:

- Names must be from 1 to 30 characters long with the following exceptions:
  - ❖ Names of databases are limited to 8 characters. Names of database links can be as long as 128 characters.
  - ❖ Names cannot contain quotation marks.
- Names are not case-sensitive.
- A name must begin with an alphabetic character from your database character set unless surrounded by double quotation marks.
- Names can only contain alphanumeric characters form your database character set and the characters_, $ and#. You are strongly discouraged from using $ and #.
- If your database character set contains multi-byte characters, it is recommended that each name for a user or a role contain at least one single-byte character.
- Names of databases links can also contain periods (.) and ampersand (&).
- Columns in the same table or view cannot have the same name. However, column in different tables or views can have the same name.

- Procedures or functions contained in the same package can have the same name, provided that their arguments are not of the same number and data types. Creating multiple procedures of functions with the same name in the same package with different arguments is called overloading the procedure or function.

Object Naming Guidelines

There are several helpful guidelines for naming objects and their parts:

- Use full, descriptive, pronounceable names (or well-known abbreviations).
- Use consistent naming rules.
- Use the same name to describe the same entity or attributes across tables.
- When naming objects, balance the objective of keeping names short and easy to use with the objective of making names as long and descriptive as possible. When in doubt, choose the more descriptive name because many people may use the objects in the database over a period of time. Your counterpart ten years from now may have difficulty in understanding a database with names like PMDD instead of PAYMENT_DUE_DATE.
- Using consistent naming rules helps users to understand the part that each table plays in your application. One such rule might be to begin the names of all tables belonging to the FINANCE application with FIN_.
- Use the same names to describe the same things across tables. For example, the department number columns of the EMP and DEPT tables should both be named DEPTNO.

3.3.4 Data Types

The data types available with SQL are given in Tables 3.1.

Table 3.1: Data Types Summary

Internal Data type	Description
VARCHAR2 (size)	Variable length character string having maximum length size bytes. Maximum size is 2000 and minimum is 1. You must specify size for a VARCHAR2.
NUMBER(p,s)	Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from 84 to 127.
LONG	Character data of variable length up to 2 gigabytes, or 231-1 bytes.
DATE	Valid data range from January 1,4712 BC to December 31, 4712 AD.
RAW(size)	Raw binary data of length size bytes. Maximum size is 255 bytes. You must specify size of a RAW value.
LONG RAW	Raw binary data of variable length up to 2 gigabytes.
ROWID(see note below)	Hexadecimal string representing the unique address of a row in its table. This data type is primarily for values returned by the ROWID pseudocolumn.
CHAR(size)	Fixed length character data of length size byte. Maximum size is 255. Default and minimum size is 1 byte.
MLSLABEL	Binary format of an operating system lable. This data type is used with Trusted Oracle7.

Character Data Types

Character data types are used to manipulate words and free-form text. These data types are used to store character (alphanumeric) data in the database character set. They are less restrictive than other data types and consequently have fewer properties. For example, character columns can store all alphanumeric values, but NUMBER columns can only store numeric values. These data types are used for character data CHAR, VARCHAR2.

CHAR Data Type

The CHAR data type specifies a fixed length character string. When you create a table with a CHAR column, you can supply the column length in bytes. RDBMS subsequently ensures that all values stored in that column have this length. If you insert a value that is shorter than the column length, RDBMS blank-pads the value to column length. If you try to insert a value that is too long for the column, RDBMS returns an error. The default for a CHAR column is 1 character and the maximum allowed is 255 characters. A zero-length string can be inserted into CHAR column, but the column is blank-padded to 1 character when used in comparisons.

- VARCHAR2 Data Type

  The VARCHAR2 data type specifies a variable length character string. When you create a VARCHAR2 column, you can supply the maximum number of bytes of data that it can hold. RDBMS subsequently stores each value in the column exactly as you specify it, provided it does not exceed the column's maximum length.

- VARCHAR Data Type

  The VARCHAR data type is currently synonymous with the VARCHAR2 data type. It is recommended that you use VARCHAR2 rather that VARCHAR. In a future version of RDBMS, VARCHAR might be a separate data type used for variable length character strings compared with different comparison semantics.

NUMBER Data Type

The NUMBER data type is used to store zero, positive and negative fixed and floating point numbers with magnitudes between 1.0×10^{-130} and 9.9×10^{125} (38 9s followed by 88 0s) with 38 digits of precision.

DATE Data Type

The DATE data type is used to store data and time information. Although data and time information can be represented in both CHAR and NUMBER data types, the DATE data type has special associated properties.

For each DATE value the following information is stored:

Century, year, month, day, hour, minute and second

To specify date value, you must convert a character or numeric value to data value with the TO_DATE function. RDBMS automatically converts character values that are in the default date format into date values when they are used in date expressions. The default date format is specified by the initialization parameter NLS_DATE_FORMAT and is a string such as 'DD-MON_YY'. This example date format includes a two-digit number for the day of the month, an abbreviation of the month name and the last two digits of the year.

If you specify a date value without a time component, the default time is 12:00 a.m. (midnight). If you specify a date value without a date, the default date is the first day of the current month. The date function SYSDATE returns the current data and time.

RAW and LONG RAW Data Types

The RAW and LONG RAW data types are used for data that is not to be interpreted (not converted when moving data between different systems) by RDBMS. These data types are intended for binary data or byte strings. For example, LONG RAW can be used to store graphics, sound, documents or areas of binary data; the interpretation is dependent on the use.

ROWID Data Type

Each row in the database has an address. You can examine a row's address by querying the pseudocolumn ROWID. Values of this pseudocolumn are hexadecimal strings representing the address of each row. These strings have the data type ROWID. You can also create tables and clusters that contain actual columns having the ROWID data type. RDBMS does not guarantee that the values of such columns are valid ROWIDs.

MLSLABEL Data Type

The MLSLABEL data type is used to store the binary format a label used on a secure operating system. Labels are used by SQL to mediate access to information. You can also define columns with this data type if you are using the standard SQL server.

Nulls

If a column or in a row has no value, then column is said to be null, or to contain a null. Nulls can appear in columns of any data type that are not restricted by NOT NULL or PRIMARY KEY integrity constraints. Use a null when the actual value is not known or when a value would not be meaningful. Do not use null to represent a value of zero, because they are not equivalent. Any arithmetic expression containing a null always evaluates to null. For example, null added to 10 is null. In fact, all operators (except concatenation) return null when given a null operand.

Tables

All data in a relational database is stored in tables. Every table has a table name and a set of columns and rows in which the data is stored. Each column is given a column name, a data type (defining characteristics of the data to be entered in the column). Usually in a relational database, some of the columns in different tables contain the same information. In this way, the tables can refer to one another.

For example, you might want to create a database containing information about the products your company manufactures. In a relational database, you can create several tables to store different pieces of information about your products, such as an inventory table, a manufacturing table and a shipping table. Each table would include columns to store data appropriate to the table (for example, the inventory table would include a column showing how much stock is on hand) and a column for the product's part number.

Views

A view is customized presentation of the data from one or more tables. Views derive their data from the tables on which they are based, which are known as base tables. All operations performed on a view actually affect the base tables of the view. You can use views for several purposes:

To give you an additional level of table security by restricting access to a predetermined set of table rows and columns. For example, you can create a view of a table that does not include sensitive data (i.e., salary information).

To hide data complexity, relational databases usually include many tables and by creating a view combining information from two or more tables, you make it easier for other users to access information from your database. For example, you might have a view that is a combination of your Employee table and Department table. A user looking at this view, which you have called emp_dept, only has to go to one place to get information, instead of having to access the two tables that make up this view.

To present the data in a different perspective from that of the base table: View provides a means to rename columns without affecting the base table. For example, to store complex queries, a query might perform extensive calculations with table information. By saving this query as a view, the calculations are performed only when the view is queried.

Indexes

An index is used to quickly retrieve information from a database project. Just as indexes help you retrieve specific information faster, a database index provides faster access to table data. Indexing creates an index file consisting of a list of records in a logical record order, along with their corresponding physical position in the table. You can use indexes to rapidly locate and display records, which is especially important with large tables, or with database composed of many tables.

Indexes are created on one or more columns of a table. Once created, an index is automatically maintained and used by the relational database. Changes to table data (such as adding new rows, or deleting rows) are automatically incorporated into all relevant indexes.

To understand how an index works, suppose you have created an employee table containing the first name, last name an employee ID number of hundreds of employees, and that you entered the name of each employee into the table as they were hired. Now, suppose you want to locate a particular record in the table. Because you entered information about each employee in no particular order, the DBMS must do a great deal of database searching to find the record.

If you create an index using the LAST-NAME column of your employee table, the DBMS has to do much less searching and can return the results of a query very quickly.

3.4 DDL AND DML COMMANDS

SQL provides a large number of commands for user-interaction. For convenience all these commands are put under three categories:

- Date Definition Language commands
- Data Manipulation Language commands
- Transaction Control commands

3.4.1 Data Definition Language Commands

Data Definition Language (DDL) commands allow users to create and/or modify various database objects that make a database. In particular they perform the following tasks:

- Create objects
- Alter or modify objects
- Drop or delete objects
- Grant and revoke privileges and roles
- Analyze information on a table or index
- Establish auditing options

The CREATE, ALTER and DROP commands require exclusive access to the object being acted upon. For example, an ALTER TABLE command fails if another user has an open transaction on the specified table.

The GRANT, REVOKE, ANALYSE, AUDIT and COMMENT commands do not require exclusive access to the object being acted upon. For example, you can analyze a table while other users are updating the table.

The following Table 3.2 shows the Data Definition Language Commands arranged alphabetically.

Table 3.2: Data Definition Language Commands

Command	Purpose
Alter Function	To recompile a stored function.
Alter Index	To redefine an index's future storage allocation.
Alter Package	To recompile a stored procedure.
Alter Procedure	To recompile a stored procedure.
Alter Profile	To add or remove a resource limit to or from a profile.
Alter Resource Cost	To specify a formula to calculate the total cost of resources used by a session.
Alter Role	To change the authorization needed to access a role.
Alter Rollback Segment	To change a rollback segment's storage characteristics, automatic refresh time, or automatic refresh mode.
Alter Snapshot Log	To change a snapshot log's storage characteristics.
Alter Table	To add a column/integrity constraint to a table. To redefine a column, to change a table's storage characteristics. To enable/disable/drop an integrity constraint. To enable/disable table locks on a table. To enable/disable all triggers on a table. To allocate an extent for the table. To allow/disallow writing to a table. To modify the degree of parallelism for a table.
Alter Tablespace	To add/rename data files. To change storage characteristics. To take a tablespace on-line/off-line. To begin/end a back up. To allow/disallow writing to a tablespace.
Alter Trigger	To enable/disable a database trigger.
Alter User	To change a user's password, default tablespace, temporary tablespace, tablespace quotas, profile, or default roles.
Alter View	To recompile a view.
Analyze	To collect performance statistics, validate structure, or identify chained rows for a table, cluster, or index.
Audit	To choose auditing for specified SQL commands or operation on schema objects.
Comment	To add a comment about a table, view, snapshot, or column to the data dictionary.
Create Control File	To recreate a control file.
Create Database	To create a database

Contd....

Create Database Link	To create a link to a remote database.
Create Function	To create a stored function.
Create Index	To create an index for a table or cluster.
Create Package	To create the specification of a stored package.
Create Package Body	To create the body of a stored package.
Create Procedure	To create a stored procedure.
Create Profile	To create a profile and specify its resource limits.
Create Role	To create a role.
Create Rollback Segment	To create a rollback segment.
Create Schema	To issue multiple CREATE TABLE, CREATE VIEW and GRANT statements in a single transaction.
Create Sequence	To create a sequence for generating sequential values.
Create Snapshot	To create a snapshot of data from one or more remote master tables.
Create Snapshot Log	To create a snapshot log containing changes made to the master table of a snapshot.
Create Synonym	To create a synonym for a schema object.
Create Table	To create a table, defining its columns, integrity constraints and storage allocation.
Create Tablespace	To create a place in the database for storage of schema objects, rollback segments and temporary segments, naming the data files to comprise the tablespace.
Create Trigger	To create a database trigger.
Create User	To create a database user.
Create View	To define a view of one or more tables or views.
Drop Cluster	To remove a cluster from the database.
Drop Database Link	To remove a database link.
Drop Function	To remove a stored function from the database.
Drop Index	To remove an index from the database.
Drop Package	To remove a stored package from the database.
Drop Procedure	To remove a stored procedure from the database.
Drop Profile	To remove a profile from the database.
Drop Role	To remove a role from the database.
Drop Sequence	To remove a sequence from the database.
Drop Snapshot	To remove a snapshot from the database
Drop Snapshot Log	To remove a snapshot log from the database.
Drop Synonym	To remove a synonym from the database.
Drop Table	To remove a table from the database.
Drop Tablespace	To remove a tablespace from the database.
Drop Trigger	To remove a trigger from the database
Drop User	To remove a user and the objects in the user's schema from the database.
Drop View	To remove a view from the database.
Grant	To grant system privileges, roles and object privileges to users and roles.
Noaudit	To disable auditing by reversing, partially or completely, the effect of a prior AUDIT statement.
Rename	To change the name of a schema object.
Revoke	To revoke system privileges, roles and object privileges from users and roles.
Truncate	To remove all rows from a table or cluster and free the space that the rows used.

3.4.2 Data Manipulation Language Commands

Data Manipulation Language (DML) commands allow users to query and manipulate data in existing schema objects. These commands implicitly commit the current transaction. These commands are listed in the following table.

Table 3.3: Data Manipulation Language Commands

Command	Purpose
DELETE	To remove rows from a table.
EXPLAIN PLAN	To return the execution plan for a SQL statement.
INSERT	To add new rows to a table.
LOCK TABLE	To lock a table or view, limiting access to it by other users.
SELECT	To select data in rows and columns from one or more tables.
UPDATE	To change data in a table.

3.4.3 Transaction Control Commands

Transaction Control Commands manage changes made by Data Manipulation Language commands. These commands are listed in the following table.

Table 3.4: Transaction Control Commands

Command	Purpose
COMMIT	To make permanent the changed made by statements issued at the beginning of a transaction.
ROLLBACK	To undo all changes since the beginning of a transaction or since a savepoint.
SAVEPOINT	To establish a point back to which you may roll.
SET TRANSACTION	To establish properties for the current transaction.

When writing SQL commands, it is important to remember a few simple rules and guidelines in order to construct valid statements that are easy to read and edit:

- SQL commands may be spread on one or many lines
- Clauses are usually placed on separate lines for enhancing readability though it is not necessary
- Tabulation can be used
- Command words cannot be split across lines
- SQL commands are not case sensitive
- An SQL command is entered at the SQL prompt. The SQL prompt acts as command line buffer. Execution takes place only when the statement is delimited by a semi-colon (;).
- Only one statement can be current at any time within the buffer and it can be run in a number of ways:
  - ❖ Place a semi-colon (;) at the end of last clause
  - ❖ Place a semi-colon/forward slash on the last line in the buffer
  - ❖ Place a forward slash at the SQL prompt
  - ❖ Issue a RUN command at the SQL prompt

Any one of the following statements is valid:

- Select * From EMP;
- Select
  *
  From
  EMP
  ;
- Select *
  FROM EMP;

3.5 RETRIEVING DATA

SQL can run through the stored tables and fetch the desired data stored therein. The command that makes this happen for the user is SELECT command. SELECT command is very powerful. In the simplest form SELECT command retrieves the data from one or more tables stored in the database. There are a number of different options that can be attached to SELECT command to retrieve data from the underlying tables. Let us juggle with some of the forms of the SELECT command.

We will assume that our database contains the following table named EMP.

Retrieving the entire Table

SELECT * FROM *tablename*;

This form of SELCT command outputs the all the columns of all the records (or rows or tuples) of the specified table (*tablename*).

Let us fire (SELECT *) SQL commands on EMP table.

SELECT * FROM emp;

This command will list the entire content of the table - EMP, as shown below.

```
SQL> select * from emp;

EMPN ENAME                JOB              MGR  DOJ              SAL       COMM DEPT
---- -------------------- ---------- ---- --------- ---------- ---------- ----
7369 VIBHOR               CLERK            7566 13-JUN-83         800          0 20
7499 ANIL                 SALESMAN         7782 15-AUG-83        1600        300 30
7521 RAKESH               SALESMAN         7698 26-MAR-84        1250        500 30
7566 KIRAN                MANAGER          7566 31-OCT-83        4500          0 20
7654 RAJAN                SALESMAN         7566 15-DEC-00        1125        800 30
7698 PRASHANT             MANAGER          7698 11-JUN-84        5000          0 30
7782 MINU                 MANAGER          7782 14-MAY-84        5500          0 10
7788 SUNIL                ANALYST          7566 05-MAR-84        3300          0 20
7844 RAMESH               SALESMAN         7782 04-JUN-84        2600       1200 30
7876 AMAN                 CLERK            7698 04-JUN-84        1575          0 20
7900 SUDHA                CLERK            7566 23-JUL-84        1600          0 30
7902 NILU                 ANALYST          7698 05-DEC-83        3400          0 20
7934 VINOD                CLERK            7566 21-NOV-83        1800          0 10

13 rows selected.

SQL> |
```

Note that the columns appear in the order in which they are stored in the original table. Besides the number of rows affected by the command are also displayed in the end (*13 rows selected*).

Retrieving Specified Columns

SELECT Col1, Col2,..... FROM *tablename*;

This form of SELECT command retrieves all the rows of only the specified columns in the order of their appearance in the command. Thus, the following command,

SELECT ename, mgr, empno FROM *emp*;

This form of SELECT command outputs the three columns of all the rows in the order given in the command, as shown below.

```
SQL> SELECT  ename, mgr, empno FROM emp;

ENAME                          MGR  EMPN
------------------------------ ---- ----
VIBHOR                         7566 7369
ANIL                           7782 7499
RAKESH                         7698 7521
KIRAN                          7566 7566
RAJAN                          7566 7654
PRASHANT                       7698 7698
MINU                           7782 7782
SUNIL                          7566 7788
RAMESH                         7782 7844
AMAN                           7698 7876
SUDHA                          7566 7900

ENAME                          MGR  EMPN
------------------------------ ---- ----
NILU                           7698 7902
VINOD                          7566 7934

13 rows selected.

SQL>
```

Retrieving Rows Satisfying a given Condition

SELECT command can be used to display only those rows that satisfy a given condition. A condition is a logical expression that results into one of the two possible values - TRUE or FALSE.

The condition is included in the SELECT command in the WHERE clause. The syntax of this form of the SELECT command is,

SELECT [*] [columns] FROM *tablename* WHERE *condition*;

For example to list all the columns of all the rows wherein SAL is more than 2000.00, the following SELECT command will be applied.

SELECT * FROM emp WHERE sal > 2000.00;

The result is shown below.

Conditional Operators

Operators Category	Operator	Meaning	Remarks
Comparison Operators	<	Is less than	
	=	Is equal to	
	>	Is greater than	
	< >	Is not equal to	
	< =	Is less than or equal to	
	> =	Is greater than or equal to	
NULL Comparison	IS NULL	Whether or not the argument is NULL	
	IS NOT NULL	Whether or not the argument is NULL	
Similarity Comparison	LIKE	Whether or not matches with the given pattern	
Range Comparison	BETWEEN	Whether or not lies between given values	
Set Inclusion	IN	Whether the value exists in the given set	

The rows selected satisfy the condition that SAL is more than 2000.00.

A conditional expression is formed using one or more logical operators. Various valid conditional (or Boolean) operators applicable in SQL are listed below. The details are discussed later.

These operators are applicable on all the built-in data types provided the values being compared are both of same data type.

Two conditional expressions can be combined to form a compound conditional expression with the help of relational operators. There are three relational operators - NOT, OR and AND. A compound condition evalues to TRUE or FALSE depending on the truth value of the operands of the relational operators. Assume that A and B are two conditional expression, then,

Truth Table of Relational Operators

AND	A	B	A AND B
	TRUE	TRUE	TRUE
	TRUE	FALSE	FALSE
	FALSE	TRUE	FALSE
	FALSE	FALSE	FALSE
OR	A	B	A OR B
	TRUE	TRUE	TRUE

Contd....

	TRUE	FALSE	TRUE
	FALSE	TRUE	FALSE
	FALSE	FALSE	FALSE
NOT	A	NOT A	
	TRUE	FALSE	
	FALSE	TRUE	

Following examples will explain this concept clearly.

Comparison Operators

1. Obtain all the rows (and all the columns) in which SAL is more than 2000 but less than 5000.

```
SQL> SELECT * FROM emp WHERE sal > 2000 AND sal < 5000;

EMPN ENAME                    JOB              MGR  DOJ              SAL       COMM DEPT
---- ------------------------ ---------- ---- --------- ---------- ---------- ----
7566 KIRAN                    MANAGER          7566 31-OCT-83        4500          0 20
7788 SUNIL                    ANALYST          7566 05-MAR-84        3300          0 20
7844 RAMESH                   SALESMAN         7782 04-JUN-84        2600       1200 30
7902 NILU                     ANALYST          7698 05-DEC-83        3400          0 20

SQL>
```

2. Obtain all the rows (and all the columns) in which SAL is either less than 5000 or equal to 5000.

```
SQL> select * from emp where sal <= 5000;

EMPN ENAME                    JOB              MGR  DOJ              SAL       COMM DEPT
---- ------------------------ ---------- ---- --------- ---------- ---------- ----
7369 VIBHOR                   CLERK            7566 13-JUN-83         800          0 20
7499 ANIL                     SALESMAN         7782 15-AUG-83        1600        300 30
7521 RAKESH                   SALESMAN         7698 26-MAR-84        1250        500 30
7566 KIRAN                    MANAGER          7566 31-OCT-83        4500          0 20
7654 RAJAN                    SALESMAN         7566 15-DEC-08        1125        800 30
7698 PRASHANT                 MANAGER          7698 11-JUN-84        5000          0 30
7788 SUNIL                    ANALYST          7566 05-MAR-84        3300          0 20
7844 RAMESH                   SALESMAN         7782 04-JUN-84        2600       1200 30
7876 AMAN                     CLERK            7698 04-JUN-84        1575          0 20
7900 SUDHA                    CLERK            7566 23-JUL-84        1600          0 30
7902 NILU                     ANALYST          7698 05-DEC-83        3400          0 20
```

The same can also be written in the following manner.

```
SQL> select * from emp where sal < 5000 or sal = 5000;

EMPN ENAME                    JOB              MGR  DOJ              SAL       COMM DEPT
---- ------------------------ ---------- ---- --------- ---------- ---------- ----
7369 VIBHOR                   CLERK            7566 13-JUN-83         800          0 20
7499 ANIL                     SALESMAN         7782 15-AUG-83        1600        300 30
7521 RAKESH                   SALESMAN         7698 26-MAR-84        1250        500 30
7566 KIRAN                    MANAGER          7566 31-OCT-83        4500          0 20
7654 RAJAN                    SALESMAN         7566 15-DEC-08        1125        800 30
7698 PRASHANT                 MANAGER          7698 11-JUN-84        5000          0 30
7788 SUNIL                    ANALYST          7566 05-MAR-84        3300          0 20
7844 RAMESH                   SALESMAN         7782 04-JUN-84        2600       1200 30
7876 AMAN                     CLERK            7698 04-JUN-84        1575          0 20
7900 SUDHA                    CLERK            7566 23-JUL-84        1600          0 30
7902 NILU                     ANALYST          7698 05-DEC-83        3400          0 20
```

In this form we have used compound conditional rather than simple conditional expression. In all the rows selected the condition is satisfied.

3. Obtain all the employees who have earned some commission.

```
SQL> select * from emp where comm <> 0;

EMPN ENAME                    JOB           MGR  DOJ              SAL       COMM DEPT
---- ------------------------ ------------- ---- --------- ---------- ---------- ----
7499 ANIL                     SALESMAN      7782 15-AUG-83        1600        300 30
7521 RAKESH                   SALESMAN      7698 26-MAR-84        1250        500 30
7654 RAJAN                    SALESMAN      7566 15-DEC-08        1125        800 30
7844 RAMESH                   SALESMAN      7782 04-JUN-84        2600       1200 30

SQL>
```

4. Obtain all the employees working in the department (30).

```
SQL> select * from emp where deptno = '30';

EMPN ENAME                    JOB           MGR  DOJ              SAL       COMM DEPT
---- ------------------------ ------------- ---- --------- ---------- ---------- ----
7499 ANIL                     SALESMAN      7782 15-AUG-83        1600        300 30
7521 RAKESH                   SALESMAN      7698 26-MAR-84        1250        500 30
7654 RAJAN                    SALESMAN      7566 15-DEC-08        1125        800 30
7698 PRASHANT                 MANAGER       7698 11-JUN-84        5000          0 30
7844 RAMESH                   SALESMAN      7782 04-JUN-84        2600       1200 30
7900 SUDHA                    CLERK         7566 23-JUL-84        1600          0 30

6 rows selected.
```

Note, since DEPTNO is char data type the comparison value should also be char type. Enclosing characters within single quotes (‘ and ‘) makes the character string char type. However, some implementations of SQL carry out limited data type conversion automatically. Thus, in ORACLE implementation of the SQL the following will yield same result.

5. Obtain all the employees who are SALESMAN and work in department No. 30.

```
SQL> select * from emp where deptno = 30;

EMPN ENAME                    JOB           MGR  DOJ              SAL       COMM DEPT
---- ------------------------ ------------- ---- --------- ---------- ---------- ----
7499 ANIL                     SALESMAN      7782 15-AUG-83        1600        300 30
7521 RAKESH                   SALESMAN      7698 26-MAR-84        1250        500 30
7654 RAJAN                    SALESMAN      7566 15-DEC-08        1125        800 30
7698 PRASHANT                 MANAGER       7698 11-JUN-84        5000          0 30
7844 RAMESH                   SALESMAN      7782 04-JUN-84        2600       1200 30
7900 SUDHA                    CLERK         7566 23-JUL-84        1600          0 30

6 rows selected.

SQL>
```

6. Obtain all the employees who are not SALESMAN and who work in department No. 30.

```
SQL> select * from emp where job='SALESMAN' and deptno='30';

EMPN ENAME                    JOB           MGR  DOJ              SAL       COMM DEPT
---- ------------------------ ------------- ---- --------- ---------- ---------- ----
7499 ANIL                     SALESMAN      7782 15-AUG-83        1600        300 30
7521 RAKESH                   SALESMAN      7698 26-MAR-84        1250        500 30
7654 RAJAN                    SALESMAN      7566 15-DEC-08        1125        800 30
7844 RAMESH                   SALESMAN      7782 04-JUN-84        2600       1200 30
```

7. Obtain all the employees whose total income (sal + comm.) is more than 1700.

```
SQL> select * from emp where sal+comm > 1700;

EMPN ENAME                JOB             MGR  DOJ              SAL       COMM DEPT
---- -------------------- --------------- ---- --------- ---------- ---------- ----
7499 ANIL                 SALESMAN        7782 15-AUG-83        1600        300 30
7521 RAKESH               SALESMAN        7698 26-MAR-84        1250        500 30
7566 KIRAN                MANAGER         7566 31-OCT-83        4500          0 20
7654 RAJAN                SALESMAN        7566 15-DEC-08        1125        800 30
7698 PRASHANT             MANAGER         7698 11-JUN-84        5000          0 30
7782 MINU                 MANAGER         7782 14-MAY-84        5500          0 10
7788 SUNIL                ANALYST         7566 05-MAR-84        3300          0 20
7844 RAMESH               SALESMAN        7782 04-JUN-84        2600       1200 30
7902 NILU                 ANALYST         7698 05-DEC-83        3400          0 20
7934 VINOD                CLERK           7566 21-NOV-83        1800          0 10

10 rows selected.
```

8. Obtain all the employees' name and total salary (sal+comm.).

```
SQL> select ename,sal+comm from emp;

ENAME                     SAL+COMM
------------------------- ----------
VIBHOR                           800
ANIL                            1900
RAKESH                          1750
KIRAN                           4500
RAJAN                           1925
PRASHANT                        5000
MINU                            5500
SUNIL                           3300
RAMESH                          3800
AMAN                            1575
SUDHA                           1600
```

Notice, that the column name is displayed as SAL+COMM. If you wish to display a more meaningful column name you can specify that as in the following.

```
SQL> select ename,sal+comm as Total from emp;

ENAME                        TOTAL
------------------------- ----------
VIBHOR                          800
ANIL                           1900
RAKESH                         1750
KIRAN                          4500
RAJAN                          1925
PRASHANT                       5000
MINU                           5500
SUNIL                          3300
RAMESH                         3800
AMAN                           1575
SUDHA                          1600
```

It is possible to include the following items in the SELECT Clause.

- Arithmetic expressions
- Column aliases
- Concatenated columns
- Literals

All these options allow the user to query data, manipulate it for query purposes; for example, performing calculations, joining columns together, or displaying literal text strings.

Arithmetic Expressions

An expression is a combination of one or more values, operators and functions, which evaluate to a value.

Arithmetic expressions may contains column names, constant numeric values and the arithmetic operators:

OPERATORS	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide

If your arithmetic expression contains more than one operator, the priority is given to *, /first, the +, - second (left to right if there are several operators with the same priority).

For example, the following command

SELECT ename, sal + 250 * 12 FROM emp;

will yield the following result.

```
SQL> SELECT ename, sal + 250 * 12 FROM emp;

ENAME                SAL+250*12
-------------------- ----------
VIBHOR                     3800
ANIL                       4600
RAKESH                     4250
KIRAN                      7500
RAJAN                      4125
PRASHANT                   8000
MINU                       8500
SUNIL                      6300
RAMESH                     5600
AMAN                       4575
SUDHA                      4600

ENAME                SAL+250*12
-------------------- ----------
NILU                       6400
VINOD                      4800

13 rows selected.
```

Column Aliases

When displaying the result of a query, SQL normally uses the selected column's name as the heading. In many cases it may be cryptic or meaningless, you can change a column's heading by using an Alias.

A column alias gives a column an alternative heading on output. Specify the alias after the column in the select list. By default, alias headings will be forced to uppercase and cannot contain blank spaces, unless the alias in enclosed in double quotes (" ").

To display the column heading ANNSAL for annual salary instead of SAL*12, use a column alias:

SELECT ename, sal * 12 ANNSAL From emp;

The result is shown below. Note this time ANNSAL is the column name instead of Sal * 12. Once defined, an alias can be used with other SQL commands.

```
SQL> SELECT ename, sal * 12 ANNSAL From emp;

ENAME                          ANNSAL
------------------------- ----------
VIBHOR                           9600
ANIL                            19200
RAKESH                          15000
KIRAN                           54000
RAJAN                           13500
PRASHANT                        60000
MINU                            66000
SUNIL                           39600
RAMESH                          31200
AMAN                            18900
SUDHA                           19200

ENAME                          ANNSAL
------------------------- ----------
NILU                            40800
VINOD                           21600

13 rows selected.
```

However, within an SQL statement, a column alias can only be used with the SELECT clause.

Literals

A literal is any character, expression, number included on the SELECT list which is not a column name or a column alias.

A literal in the SELECT list is output for each row returned. Literal strings of free formal text can be included in the query result and are treated like a column in the select list.

The following statement contains literal selected with concatenation and a column alias:

SELECT EMPLOYEE ENAME, '-', 'Works in department-', DEPTNO FROM EMP;

The result is shown below.

EMPLOYEE
VIBHOR-Works in department-20
ANIL-Works in department-30
RAKESH-Works in department-30
KIRAN-Works in department-20
RAJAN-Works in department-30
PRASHANT-Works in department-30
MINU-Works in department-10
SUNIL-Works in department-20
RAMESH-Works in department-30
AMAN- Works in department-20
SUDHA-Works in department-30
NILU-Works in department-20
VINOD-Works in department-10

Handling Null Values

If a row lacks a data value for a particular column, that value is said to be NULL. A null value is a value, which is either unavailable, unassigned, unknown or inapplicable. A null value is not the same as zero. Zero is a number. Null values take up one byte of internal 'storage' overhead. Null Values are Handled Correctly by SQL.

If any column value in an expression is null, the result is null. In the following statement, only Salesmen have a remuneration result:

SELECT ENAME, SAL * 12 + COMM ANNUAL_SAL FROM EMP;

ENAME	ANNUAL_SAL
VIBHOR	
ANIL	19500
RAKESH	15500
KIRAN	
RAJAN	16400
PRASHANT	
MINU	
SUNIL	
KING	
RAMESH	18000
AMAN	
SUDHA	
NILU	
VINOD	

In order to achieve a result for all employees, it is necessary to convert the null value to a number. We use the NVL function to convert a null value to a non-null value.

Use the NVL function to convert null values from the previous statement to zero.

SELECT ENAME, SAL*12 + NVL(COMM, 0) ANNUAL_SAL FROM EMP;

The result is shown below.

ENAME	ANNUAL_SAL
VIBHOR	9600
ANIL	19500
RAKESH	155500
KIRAN	35700
RAJAN	16400
PRASHANT	34200
MINU	29400
SUNIL	36000
KING	60000

Contd.....

RAMESH	18000
AMAN	13200
SUDHA	11400
NILU	36000
MILLER	15600

NVL expects two arguments – an expression and a non-null value. Note that you can use the NVL function to convert a null number, date or even character string to another number, date or character string, as long as the data types match.

NVL (Date column, '01-jan-88')

NVL (Number column, 9)

NVL (char column, 'string')

Preventing the Selection of Duplicate Rows

Unless you indicate otherwise, SQL displays the result of query without eliminating duplicate entries. For instance, the following query,

SELECT DEPTNO FROM EMP;

produces the following result.

DEPTNO
20
30
30
20
30
30
10
20
10
30
20
30
20
10

To eliminate duplicate values in the result, include the DISTINCT qualifier in the SELECTcommand as follows.

SELECT DISTINCT DEPTNO FROM EMP;

This time the result will not contain duplicate values.

DEPTNO
20
30
10

Multiple columns may be specified after the DISTINCT qualifier and the DISTINCT affects all selected columns.

To display distinct values of DEPTNO and JOB, enter:

```
SELECT DISTINCT DEPTNO, JOB FROM EMP;
```

The result is given below.

DEPTNO	JOB
10	CLERK
10	MANAGER
10	PRESIDENT
20	ANALYST
20	CLERK
20	MANAGER
30	CLERK
30	MANAGER
30	SALESMAN

Ordered by Clause

This displays a list of all different combinations of jobs and department numbers. The order of rows returned in a query result is undefined. The ORDER BY clause may be used to sort the rows. If used, ORDER BY must always be the last clause in the SELECT statement.

To sort by ENAME, enter:

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

The result is shown below.

ENAME	JOB	SAL	DEPTNO
AMAN	CLERK	1,100.00	20
ANIL	SALESMAN	1,600.00	30
KIRAN	MANAGER	2,975.00	20
MINU	MANAGER	2,450.00	10
NILU	ANALYST	3,000.00	20
PRASHANT	MANAGER	2,850.00	30
RAJAN	SALESMAN	1,25.00	30
RAKESH	SALESMAN	1,250.00	30
RAMESH	SALESMAN	1,500.00	30
SUDHA	CLERK	950	30
SUNIL	ANALYST	3,000.00	20
VIBHOR	CLERK	800	20
VINOD	CLERK	1,300.00	10

The default sort order is ASCENDING which defines the following sort order.

- Numeric values lowest first
- Date values earliest first
- Character values alphabetically (a to z)

To reverse the order, the command word DESC is specified after the column name in the ORDER BY clause.

To reverse the order of the DOJ column, so that the latest dates are displayed first, enter:

SELECT ENAME, JOB, DOJ FROM EMP ORDER BY DOJ DESC;

ENAME	JOB	DOJ
SUDHA	CLERK	23-Jul-84
PRASHANT	MANAGER	11-Jun-84
RAMESH	SALESMAN	4-Jun-84
AMAN	CLERK	4-Jun-84
MINU	MANAGER	14-May-84
RAKESH	SALESMAN	26-Mar-84
SUNIL	ANALYST	5-Mar-84
RAJAN	SALESMAN	5-Dec-83
NILU	ANALYST	5-Dec-83
VINOD	CLERK	21-Nov-83
KIRAN	MANAGER	31-Oct-83
ANIL	SALESMAN	15-Aug-83
VIBHOR	CLERK	13-Jun-83

It is possible to ORDER BY more than one column. The limit is the number of columns on the table. In the ORDER BY clause, specify the columns to order by separated commas. If any or all are to be reversed, specify DESC after any or each column.

To order by two columns and display in reverse order of salary, enter:

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

The result is shown below.

DEPTNO	ENAME	JOB	SAL
10	MINU	MANAGER	2,450.00
10	VINOD	CLERK	1,300.00
20	SUNIL	ANALYST	3,000.00
20	NILU	ANALYST	3,000.00
20	KIRAN	MANAGER	2,975.00
20	AMAN	CLERK	1,100.00
20	VIBHOR	CLERK	800
30	RAJAN	SALESMAN	1,25.00
30	PRASHANT	MANAGER	2,850.00
30	ANIL	SALESMAN	1,600.00
30	RAMESH	SALESMAN	1,500.00
30	RAKESH	SALESMAN	1,250.00
30	SUDHA	CLERK	950

Aggregate Function and Group By Clause

An aggregate function is a that computes values based on more than one row. In these cases GROUPBY clause can be used to display the value of aggregate function of a group of some specified column. SQL has following aggregate functions. They are:

AVG: Returns average of the group

COUNT: Returns the number of rows in a group

MAX: Returns maximum value of a column in a group

MIN: Returns minimum value of a column in a group

SUM: Returns the sum of a numeric column of a group

The common syntax of these functions is:

SELECT function_name(column_name);

Here, function_name is one of the functions listed above and the column_name is a numeric (or non-numeric) column name, whichever is applicable.

For example, to compute the total salary of all the clerks in our database, the following SQL query will be used.

GROUP BY clause can be used to obtain aggregate values in a group of rows. Or instance, to find the salaries of all the job types, use the following SQL query:

SELECT JOB, SUM(SAL) FROM EMP GROUP BY JOB;

Multi-table Queries

SQL can retrieve rows and columns from more than one table. However, it makes sense only when the tables are related to each other with a key or more keys. We will take the following tables to illustrate multi-table queries.

Table 3.5: Employee

EmpId	EmpName	EmpAddr	EmpSal	EmpDOJ
010	Vibhor	A-56, Naraina, New Delhi	6000	12/11/2005
099	Prashant	D-11/C, Uttam Nagar, Mumbai	7000	01/01/2005
011	Minu Prasad	11, Janak Puri, Chennai	6600	02/11/2005
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	01/05/2003
111	Pankaj Sharma	23, Lado Sarai, Sitamarhi, Bihar	8000	23/04/2004

Table 3.6: Projects

ProjId	EmpId	Perks
111	010	3000
112	111	6000
117	099	16000
114	011	1000
101	100	4000
111	099	3000
112	100	6000
117	010	16000
114	010	1000
111	100	4000

Table 3.7: ProjectLocation

ProjId	Location
111	New Delhi
112	London
117	Vienna
114	Kolkata
101	Bihar

A simple SELECT statement is the most basic way to query multiple tables. You can call more than one table in the FROM clause to combine results from multiple tables. Here's an example of how this works:

```
SELECT table1.column1, table2.column2 FROM table1, table2 WHERE
table1.column1 = table2.column1;
```

Suppose we wish to list all the employees who worked on the project with PROJID='111". The desired information is stored in the two tables - Employee and Projects. The required SELECT query is,

```
SELECT Employee.EmpName, Projects.ProjId FROM Employee, Projects
WHERE Employee.EmpId=Projects.EmpId;
```

The output is shown below.

EmpName	ProjId
Vibhor	117
Vibhor	101
Vibhor	114
Minu Prasad	111
Prashant	111
Prashant	112
Rajan Wadhwa	112
Rajan Wadhwa	114
Rajan Wadhwa	117
Panka Sharma	111

JOIN Statements

JOIN operation virtually combines two tables on specified columns and treats the resulting view as a single table. There are three varieties of JOIN.

INNER JOIN: Combines the rows from the two mentioned tables where the specified fields are equal.

LEFT JOIN: Takes all the rows from the Table1 and only those rows from Table2 where the column values are equal.

RIGHT JOIN: Takes all the rows from the Table2 and only those rows from Table1 where the column values are equal.

For example, the following SQL will return the given rows.

```
SELECT Employee.EmpId, Employee.EmpName, Employee.EmpAddr,
Employee.EmpSal, Projects.ProjId, Projects.Perks
```

FROM Projects INNER JOIN Employee ON Projects.EmpId = Employee.EmpId;

EmpId	EmpName	EmpAddr	EmpSal	ProjId	Perks
010	Vibhor	A-56, Naraina, New Delhi	6000	117	16000
010	Vibhor	A-56, Naraina, New Delhi	6000	101	4000
010	Vibhor	A-56, Naraina, New Delhi	6000	114	1000
011	Minu Prasad	11, Janak Puri, Chennai	6600	111	4000
099	Prashant	D-11/C, Uttam Nagar, Mumbai	7000	111	3000
099	Prashant	D-11/C, Uttam Nagar, Mumbai	7000	112	6000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	112	6000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	114	1000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	117	16000
111	Panka Sharma	23, Lado Sarai, Sitamarhi, Bihar	8000	111	3000

SELECT Employee.EmpId, Employee.EmpName, Employee.EmpAddr, Employee.EmpSal, Projects.ProjId, Projects.Perks

FROM Projects LEFT JOIN Employee ON Projects.EmpId = Employee.EmpId;

EmpId	EmpName	EmpAddr	EmpSal	ProjId	Perks
010	Vibhor	A-56, Naraina, New Delhi	6000	117	16000
010	Vibhor	A-56, Naraina, New Delhi	6000	101	4000
010	Vibhor	A-56, Naraina, New Delhi	6000	114	1000
011	Minu Prasad	11, Janak Puri, Chennai	6600	111	4000
099	Prashant	D-11/C, Uttam Nagar, Mumbai	7000	111	3000
099	Prashant	D-11/C, Uttam Nagar, Mumbai	7000	112	6000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	112	6000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	114	1000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	117	16000
111	Panka Sharma	23, Lado Sarai, Sitamarhi, Bihar	8000	111	3000

SELECT Employee.EmpId, Employee.EmpName, Employee.EmpAddr, Employee.EmpSal, Projects.ProjId, Projects.Perks

FROM Projects RIGHT JOIN Employee ON Projects.EmpId = Employee.EmpId;

EmpId	EmpName	EmpAddr	EmpSal	ProjId	Perks
010	Vibhor	A-56, Naraina, New Delhi	6000	117	16000
010	Vibhor	A-56, Naraina, New Delhi	6000	101	4000
010	Vibhor	A-56, Naraina, New Delhi	6000	114	1000
011	Minu Prasad	11, Janak Puri, Chennai	6600	111	4000
099	Prashant	D-11/C, Uttam Nagar, Mumbai	7000	111	3000
099	Prashant	D-11/C, Uttam Nagar, Mumbai	7000	112	6000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	112	6000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	114	1000
100	Rajan Wadhwa	1, Safadar Jung, New Delhi	10000	117	16000
111	Panka Sharma	23, Lado Sarai, Sitamarhi, Bihar	8000	111	3000

Drop Constraint

Sometimes it is required to remove a constraint defined on a table or on its columns. For this purpose DROP CONSTRAINT clause is used in the ALTER TABLE SQL statement. The syntax is,

ALTER TABLE table_name DROP CONSTRAINT constraint_name

DROP CONSTRAINT is used to drop a named constraint from a table. For example, create a table named -student - as described below.

student	RN	Name	Class	Section
	100	Sachin	VII	B
	200	Rahul	VIII	D

Suppose we put a constraint on the table as - "No roll number (RN) must be repeated. The constraint on the RN column is UNIUE constraint. The corresponding SQL command to create this table is,

CREATE TABLE student (rn char(3),name varchar(25), class varchar(4), section char(1));

Now, let us add a constraint (we will call it CRN) that does not allow any duplicate entry into the RN column. The required SQL statement is,

ALTER TABLE student ADD CONSTRAINT crn UNIQUE (RN);

Now, let us insert some rows into this table.

```
SQL> insert into student values('001','Vibhor','IX','A');

1 row created.

SQL> insert into student values('001','Mahesh','VII','C');

insert into student values('001','Mahesh','VII','C')
*

ERROR at line 1:

ORA-00001: unique constraint (MANOJ.CRN) violated
```

Since we tried to insert '001' in the RN column in which '001' is already stored, the error was generated because UNIQUE constraint (crn) defined on this column.

The DROP CONSTRAINT clause can be used to modify the table so that RN column may allow duplicate entries. The command is,

ALTER TABLE student DROP CONSTRAINT crn;

```
SQL> alter table student drop constraint crn;

Table altered.
```

Now if you insert the duplicate value in this column no error will result.

```
SQL> insert into student values('001','Mahesh','VII','C');

1 row created.
```

```
SQL> select * from student;
RN NAME                CLAS S
--- -------------------- ---- -
001 Vibhor             IX  A
001 Mahesh             VII C
```

Commit, Rollback and Savepoint

These three SQL commands are used in transaction processing. COMMIT command makes all the changes made to the database since the last COMMIT. ROLLBACK command undoes all the changes made to the database to a specified point and SAVEPOINT allows the users to define a label in the transaction sequence where commit and rollback can be effected.

For example, consider the following transaction.

```
SQL> insert into student values ('002','Ganesh','X','D');
1 row created.
SQL> Insert into student values ('003','Dinesh','XI','E');
1 row created.
SQL> insert into student values ('005','Ramesh','XII','A');
1 row created.
```

Here, establish a savepoint named FirstSavePoint. The command is,

```
SQL>savepoint FirstSavePoint;
Savepoint created
```

Insert three more rows:

```
SQL> insert into student values ('010','Suresh','IX','D');
1 row created.
SQL> insert into student values ('011','Sarvesh','VI','A');
1 row created.
SQL> insert into student values ('012','Rakesh','XII','D');
1 row created.
```

Establish a second savepoint named SecondSavePoint.

```
SQL>savepoint SecondSavePoint;
Savepoint created
```

Again insert some rows.

```
SQL> insert into student values ('021','Anil','IX','D');
1 row created.
```

```
SQL> insert into student values ('022','Sunil','IX','A');
1 row created.
```

Let's check what we have entered.

```
SQL> select * from student;
RN    NAME          CLAS S
--- ------------------------ ---- -
001   Vibhor        IX   A
001   Mahesh        VII  C
002   Ganesh        X    D
003   Dinesh        XI   E
005   Ramesh        XII  A
010   Suresh        IX   D
011   Sarvesh       VI   A
012   Rakesh        XII  D
021   Anil          IX   D
022   Sunil         IX   A
10 rows selected.
```

Now, rolling back to savepoint SecondSavePoint.

```
SQL> rollback to SecondSavePoint;
Rollback complete.
```

Checking once more the table's content:

```
SQL> select * from student;
RN  NAME      CLAS S
--- ------------------------ ---- -
001 Vibhor    IX   A
001 Mahesh    VII  C
002 Ganesh    X    D
003 Dinesh    XI   E
005 Ramesh    XII  A
010 Suresh    IX   D
```

011 Sarvesh VI A

012 Rakesh XII D

8 rows selected.

All DMLS (data manipulated) after savepoint SecondSavePoint have been rolled back. The changes are thus not committed unless either the database is closed or an explicit commit is executed.

SQL> COMMIT;

The BETWEEN Operator

It tests for values between and inclusive of low and high range. Suppose we want to see those employees whose salary is between 1000 and 2000. Enter the following SQL query.

SELECT ENAME, SAL FROM EMP WHERE SAL BETWEEN 1000 AND 2000;

The result is shown below.

ENAME	SAL
ANIL	1,600.00
RAKESH	1,250.00
RAMESH	1,500.00
AMAN	1,100.00
VINOD	1,300.00

Note that values specified are inclusive and the lower limit must be specified first.

The IN Operator

It tests for values in a specified list. To find all employees who have one of the three MGR numbers - 7902, 7566 and 7788 - , enter the following query.

SELECT ENAME, SAL, MGR FROM EMP WHERE MGR IN (7902, 7566, 7788);

The result is shown below.

ENAME	MGR	SAL
VIBHOR	7902	800.00
SUNIL	7566	3,000.00
AMAN	7788	1,100.00
NILU	7566	3,000.00

If character or dates are used in the list they must be enclosed in single quotes (' ').

The Like Operator

Sometimes you may not know the exact value to search for. Using the LIKE operator, it is possible to select rows that match a character pattern. The character pattern matching operation may be referred to as 'wild-card' search. Two symbols can be used to construct the search string.

Symbol	Represents
%	Any sequence of zero or more characters
_	Any single character

For example, to list all employees whose name starts with an S, enter:

SELECT * FROM EMP WHERE ENAME LIKE 'S%';

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
778	SUNIL	ANALYST	7566	5-Mar-84	3,000.00		20
7900	SUDHA	CLERK	7698	23-Jul-84	950.00		30

This can be used to search for a specific number of characters. For example, to list all employees who have a name exactly 4 characters in length, enter:

SELECT * FROM EMP WHERE ENAME LIKE '____';

The result is shown below.

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7499	ANIL	SALESMAN	7698	15-Aug-83	1,600.00	300	30
7782	MINU	MANAGER	7839	14-May-84	2,450.00		10
7876	AMAN	CLERK	7788	4-Jun-84	1,100.00		20
7902	NILU	ANALYST	7566	5-Dec-83	3,000.00		20

The % and _ may be used in any combination with literal characters.

IS NULL Operator

The IS NULL operator specifically tests for values that are NULL. To find all employees who have no commission, you are testing for a NULL.

SELECT * FROM EMP WHERE COMM IS NULL;

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7369	VIBHOR	CLERK	7902	13-Jun-83	800		20
7566	KIRAN	MANAGER	7839	31-Oct-83	2,975.00		20
7698	PRASHANT	MANAGER	7839	11-Jun-84	2,850.00		30
7782	MINU	MANAGER	7839	14-May-84	2,450.00		10
7788	SUNIL	ANALYST	7566	5-Mar-84	3,000.00		20
7876	AMAN	CLERK	7788	4-Jun-84	1,100.00		20
7900	SUDHA	CLERK	7698	23-Jul-84	950		30
7902	NILU	ANALYST	7566	5-Dec-83	3,000.00		20
7934	VINOD	CLERK	7782	21-Nov-83	1,300.00		10

Negating Expressions

The following operators negate the test condition:

OPERATOR	DESCRIPTION
!=	not equal to (VAX, UNIX, PC)
^=	not equal to (IBM)
< >	not equal to (all O/S)
NOT COLNAME=	not equal to

Contd....

NOT COLNAME>	not greater than
NOT BETWEEN	not between two given values
NOT LIKE	not in given list of values
IS NOT NULL	Is not null value

To find all the employees whose salary is not between 1000 and 2000, enter the following query.

SELECT * FROM EMP WHERE SAL NOT BETWEEN 1000 AND 2000;

The result is given below.

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7369	VIBHOR	CLERK	7902	13-Jun-83	800		20
7566	KIRAN	MANAGER	7839	31-Oct-83	2,975.00		20
7698	PRASHANT	MANAGER	7839	11-Jun-84	2,850.00		30
7782	MINU	MANAGER	7839	14-May-84	2,450.00		10
7788	SUNIL	ANALYST	7566	5-Mar-84	3,000.00		20
7902	NILU	ANALYST	7566	5-Dec-83	3,000.00		20

To find those employees whose job does not start with A, enter the following query. SELECT * FROM EMP WHERE JOB NOT LIKE 'A%';

The result is shown below.

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7369	VIBHOR	CLERK	7902	13-Jun-83	800		20
7521	RAKESH	SALESMAN	7698	26-Mar-84	1,250.00	500	30
7566	KIRAN	MANAGER	7839	31-Oct-83	2,975.00		20
7654	RAJAN	SALESMAN	7698	5-Dec-83	1,25.00	1,400.00	30
7698	PRASHANT	MANAGER	7839	11-Jun-84	2,850.00		30
7782	MINU	MANAGER	7839	14-May-84	2,450.00		10
7788	SUNIL	ANALYST	7566	5-Mar-84	3,000.00		20
7844	RAMESH	SALESMAN	7698	4-Jun-84	1,500.00	0	30
7900	SUDHA	CLERK	7698	23-Jul-84	950		30
7902	NILU	ANALYST	7566	5-Dec-83	3,000.00		20
7934	VINOD	CLERK	7782	21-Nov-83	1,300.00		10

To find all employees who have earned a commission, enter the following query.

SELECT * FROM EMP WHERE COMM IS NOT NULL;

The result is shown below.

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7499	ANIL	SALESMAN	7698	15-Aug-83	1,600.00	300	30
7521	RAKESH	SALESMAN	7698	26-Mar-84	1,250.00	500	30
7654	RAJAN	SALESMAN	7698	5-Dec-83	1,25.00	1,400.00	30
7844	RAMESH	SALESMAN	7698	4-Jun-84	1,500.00	0	30

Please note the following:

- If a NULL value is used in a comparison, then the comparison operator should be either IS or IS NOT NULL. If these operators are not used and NULL values are compared, the result is always FALSE.
- For example, COMM != NULL is always FALSE. The result is false because a NULL value can neither be either equal or unequal to any other value, even another NULL. Note that an error is not raised; the result is simply always false.

Querying Data with Multiple Conditions

The AND an OR operators may be used to make compound logical expressions. The AND predicate will expect both conditions to be 'true'; whereas the OR predicate will expect either condition to be 'true.

In the following two examples the conditions are the same, the predicate is different. See how the result is dramatically changed.

To find all clerks who earn between 1000 and 2000, enter:

SELECT * FROM EMP WHERE SAL BETWEEN 1000 AND 2000 AND JOB='CLERK';

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7876	AMAN	CLERK	7788	4-Jun-84	1,100.00		20
7934	VINOD	CLERK	7782	21-Nov-83	1,300.00		10

To find all employees who are either clerks or all employees who earn between 1000 and 2000, enter:

SELECT * FROM EMP WHERE SAL BETWEEN 1000 AND 2000 OR JOB='CLERK';

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7369	VIBHOR	CLERK	7902	13-Jun-83	800		20
7499	ANIL.	SALESMAN	7698	15-Aug-83	1,600.00	300	30
7521	RAKESH	SALESMAN	7698	26-Mar-84	1,250.00	500	30
7844	RAMESH	SALESMAN	7698	4-Jun-84	1,500.00	0	30
7876	AMAN	CLERK	7788	4-Jun-84	1,100.00		20
7900	SUDHA	CLERK	7698	23-Jul-84	950		30
7934	VINOD	CLERK	7782	21-Nov-83	1,300.00		10

You may combine AND and OR in the same logical expression. When AND and OR appear in the same WHERE clause, all the ANDs are performed first, then all the ORs are performed. We say that AND has a higher precedence than OR.

Since AND has a higher precedence than OR, the following SQL statement returns all managers with salaries over 1500 and all salesmen.

```
SELECT * FROM EMP
WHERE  SAL> 2500  AND  JOB='MANAGER' OR JOB= 'SALESMAN';
```

EMPNO	ENAME	JOB	MGR	DOJ	SAL	COMM	DEPTNO
7499	ANIL	SALESMAN	7698	15-Aug-83	1,600.00	300	30
7521	RAKESH	SALESMAN	7698	26-Mar-84	1,250.00	500	30
7566	KIRAN	MANAGER	7839	31-Oct-83	2,975.00		20
7654	RAJAN	SALESMAN	7698	5-Dec-83	1,25.00	1,400.00	30
7698	PRASHANT	MANAGER	7839	11-Jun-84	2,850.00		30
7844	RAMESH	SALESMAN	7698	4-Jun-84	1,500.00	0	30

If you wanted to select all managers and salesman with salaries over 1500, you would enter:

```
SELECT *  FROM EMP
WHERE SAL  >  2500 AND (JOB='MANAGER' OR JOB='SALESMAN');
```

The parentheses specify the order in which the operators should be evaluated. In the second example, the OR operator is evaluated before the AND operator.

Inserting Records

To insert a record into a table, INSERT command can be used. The syntax of this command is:

```
INSERT INTO <tablename>
VALUES (value1, value 2,.....);
```

For example, the following query inserts a record into the table EMP.

```
INSERT INTO emp
VALUES ('101', 'Nandi', 'President', '17-NOV-88', 5000, null, '10');
```

To insert values into only EMPNO, DEPTNO and ENAME fields, enter the following query.

```
INSERT INTO emp (empno, deptno, ename)
VALUES ('101', '29', 'Sujit');
```

Updating a Table

To change the values of the field in specified table, UPDATE command can be used. The syntax is:

```
UPDATE <tablename>
SET column1= expression, column2= expression
WHERE condition;
```

For example, to change the salary of 'VIBHOR' to 5000, enter the following query.

```
UPDATE emp
SET salary = 5000
WHERE ename = 'VIBHOR';
```

Be careful with this query. If *where* clause is omitted all rows are updated.

Delete

To remove one or more row from a table DELETE query is used. The syntax is:

```
Delete From <tablename>
Where<condition>
```

For example, to delete all the rows whose salary is more than 1000, enter the following query.

```
DELETE FROM emp
WHERE sal > 1000;
```

If the WHERE clause is omitted all the rows of the specified table will be deleted. A part of the row cannot be deleted.

3.6 DATA DEFINITION LANGUAGE

Data definition language is used to create, alter or remove a data structure, table or Database Structure.

In an RDBMS data is stored in data structures known as tables, comprising of rows and columns. A table is created in a logical unit of the database called table space. A database may have one or more table space.

The table space is divided into segments, which is a set of database blocks allocated for storage of database structures, namely tables, indexes etc.

Segments are defined using storage parameters, which in turn are expressed in terms of extents of data. An extent is an allocation of database space which itself contains many blocks - the basic unit of storage.

3.6.1 Creating a Table

To create a TABLE Structure use CREATE TABLE query whose syntax is given below.

```
CREATE TABLE <tablename>
Column1 data type(size) [null/not null]
column2 data type (size),.....)
```

For example, to create the table EMP enter the following query.

```
Create table emp
(       empo number (4) not null
        ename varchar 2 (10),
        job varchar2 (9),
        DOJ date,
        sal number (7,2),
        comm number (7,2),
        deptno number (2 not null)
);
```

Alter Table

ALTER is used to change the structure of an existing table. The syntax is:

```
ALTER TABEL TABLENAME
    [ADD column_element....., MODIFY]
```

For example, to insert a column called NET_SAL as a number type into EMP table, use the following query.

```
ALTER TABLE EMP
    ADD (NET_SAL NUMBER (10));
```

It is not possible to change the name of an existing column or delete an existing column. The data type of an existing column can be changed, if the field is blank for all existing View.

A view is a logical (Virtual) table derived from one or more base tables or views. It is basically a subschema defined as a subset of the Schema. Views are like windows through which one can view information stored in tables. View does not contain data of its own. It is stored as a query. The contents are taken from other tables through the execution of the query. As the contents of the table change, the view would also change dynamically. The syntax to create a view is given below.

```
CREATE VIEW <view name>
    AS <query>;
```

One may UPDATE and DELETE rows in a view, based on a single table and its query does not contain GROUP BY clause the DISTINCT clause.

One may INSERT rows if the views observe the same restrictions and its query contains on columns defined by expressions.

For example, in order to create a view of EMP table named DEPT20, to show the employees in department 20 and their annual salary use the following command.

```
CREATE VIEW dept20
AS SELECT ename, sal *12 FROM emp WHERE departno= 20;
```

Once the VIEW is created, it can be treated like any other table. Thus the following is a valid command.

```
SELECT * from dept20;
```

Create Sequence

A sequence is a database object that generates unique integer values each time it is referred to. CREATE SEQUENCE command creates a sequence. The syntax is given below.

```
CREATE SEQUENCE seq_name
        [INCREMENTED BY n]
        [START WITH n]
        [MAXVALUE n]
        [MINIVALUE n];
```

START WITH clause specifies the initial value of the sequence; INCREMENTED BY clause specifies the values that must be added to the previous value to get the new value; and MAXVALUE and MINVALUE specify the maximum and the minimum values respectively that the sequence can generate.

For example,

```
CREATE SEQUENCE my_seq
        INCREMENTED BY 10
        START WITH 1
        MAXVALUE 100;
```

will create a sequence named my_seq whose first value will be 1, next 11, next 21, and so on. The maximum value generated by this sequence will be less than or equal to 100.

Create Index

To create an index on one or more column of a table or a cluster, CREATE command is used. The syntax is given below.

```
CREATE [UNIQUE] INDEX index_name
        ON table_name
```

(column_name [, column_name...])

TABLESPACE tablespace;

For example,

CREATE INDEX empIndex ON employee(ename) TABLESPACE company;

will create an index an index named empIndex on the ename column of table employee of company tablespace. More than one column can be used for indexing by specifying the comma-separated list of the columns in that order.

Check Your Progress

Fill in the blanks:

1. Data Definition Language, is a part of the SQL language used to define data and ..... in a database.
2. Embedded SQL refers to the use of standard ..... commands embedded within a procedural programming language.
3. Character data types are used to manipulate words and .....
4. If a row lacks a ..... for a particular column, that value is said to be NULL.

3.7 LET US SUM UP

SQL is the set of commands that programs and users may use to access data within the database that supports it. SQL is a non-procedural language in that the users do not have to specify how the SQL commands are to be executed. Embedded SQL refers to the use of standard SQL commands embedded within a procedural programming language. A schema is a collection of logical structures of data, of schema objects. A schema is owned by a database user and has the same name as that user. Character data types are used to manipulate words and free-form text. These data types are used to store character (alphanumeric) data in the database character set. The VARCHAR2 data type specifies a variable length character string. The NUMBER data type is used to store zero, positive and negative fixed and floating point numbers with magnitudes between $1.0x10^{-130}$ and $9.9x10^{125}$ (38 9s followed by 88 0s) with 38 digits of precision. The DATE data type is used to store data and time information. The RAW and LONG RAW data types are used for data that is not to be interpreted by RDBMS. Data Definition Language (DDL) commands allow users to create and/or modify various database objects that make a database. Data Manipulation Language (DML) commands allow users to query and manipulate data in existing schema objects.

3.8 KEYWORDS

SQL (Structured Query Language): The set of commands that programs and users may use to access data within the database that supports it.

Non-procedural Language: A language which is free from writing algorithms.

Schema Objects: A collection of logical structures of data.

View: A customized presentation of the data from one or more tables.

DDL (Data Definition Language): Commands that allow users to create and/or modify various database objects that make a database.

DML (Data Manipulation Language): Commands allow users to query and manipulate data in existing schema objects.

Sequence: A database object that generates unique integer values each time it is referred to.

3.9 QUESTIONS FOR DISCUSSION

1. Suppose we have a table having structure like employee

   (emp_id number (3), emp_name varchar2 (15), dep_no number (2), emp_desig varchar2 (5), salary number (8.2))

   (a) Create a table named employee as given above.

   (b) Insert <1, 'Mohan', '01,'manager', 25000.00> into employee.

   (c) Insert <2, 'Ram', '02,'clerk', 5000.00> into employee.

   (d) Insert <3, ' Ramesh,' 03', 'Accountant', 7000.00> into employee.

   (e) Insert <4, ' Rajesh,' 05', 'clerk', 500000.00> into employee.

   Now make the following queries-

   (f) Find the employee whose designation is manager.

   (g) Find the employee's details whose salary is second longest.

   (h) Find the name of all employees who are clerks.

   (i) Find the employees who belong to the same department.

   (j) Update Ramesh department no to 04.

2. What do you understand by SQL?
3. What we view in DBMS?
4. Why do we use indexes?
5. What do you understand by DDL?
6. Make a list of commands used in DDL.
7. What do you understand by DML?
8. What are the uses of Insert, Delete and Update commands?
9. Why do we use select statement?
10. What is the function of Create, Alter commands?

Check Your Progress: Model Answers

1. Objects
2. SQL
3. free-form text
4. data value

3.10 SUGGESTED READINGS

Peter Rob, Carlos Coronel, *Database Systems: Design, Implementation and Management*, Seventh edition, Thomson Learning, 2007

Silberschatz, Korth, Sudarshan, *Database System Concepts*, Fifth edition, McGraw-Hill, 2005

Elmasari Navathe, *Fundamentals of Database Systems*, Third edition, Pearson Education Asia, 2001

E. J. Yannakoudakis, *The Architectural Logic of Database Systems*, Springer-Verlag, Digitized 2007

Fred R. McFadden, Jeffrey A. Hoffer, *Database Management*, Benjamin/Cummings, Digitized 2007

Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems*, Third edition, McGraw-Hill Higher Education, 2003

LESSON

4

SQL*PLUS

CONTENTS

4.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of entering and executing commands
- Discuss SQL command syntax
- Describe the running of SQL*PLUS commands
- Identify and explain the SQL*PLUS functions

4.1 INTRODUCTION

Unless stated otherwise, descriptions of command use are generally applicable to both command-line and iSQL*Plus user interfaces. In command-line SQL*Plus, you type commands at the SQL*Plus prompt. Usually, you separate the words in a command from each other by a space or tab. You can use additional spaces or tabs between words to make your commands more readable.

4.2 ENTERING AND EXECUTING COMMANDS

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 in either the command-line or the iSQL*Plus user interfaces:

- 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.

You can use the Backspace and the Delete keys in both command-line SQL*Plus and iSQL*Plus. In iSQL*Plus, you can cut and paste using your web browser's edit keys to edit the statements in the Input area. You can also cut or copy scripts or statements from other applications such as text editors, and paste them directly into the Input area.

In iSQL*Plus, the Save Script button enables you to save scripts to a text file. You can also load scripts with the Load Script button. Saving and loading scripts may be useful when editing and testing.

4.2.1 The SQL Buffer

The area where SQL*Plus stores your most recently entered SQL command or PL/SQL block (but not SQL*Plus commands) is called the SQL buffer. The command or block remains there until you enter another. If you want to edit or re-run the current SQL command or PL/SQL block, you may do so without re-entering it.

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

4.2.2 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. In iSQL*Plus, you type a command or a script into the Input area and click the Execute button to execute the contents of the Input area. The results of your script are displayed below the Input area by default. Use the History screen to access and rerun commands previously executed in the current session. iSQL*Plus executes a SQL or PL/SQL statement at the end of the Input area, even if it is incomplete or does not have a final ";" or "/". If you intend to run iSQL*Plus scripts in the SQL*Plus command-line, you should make sure you use a ";" or "/" to terminate your statements.

iSQL*Plus retains the state of your current system variables and other options from one execution to the next. If you use the History screen to re-execute a script, you may get different results from those previously obtained, depending on the current system variable values. Some SQL*Plus commands have no logical sense or are not applicable in iSQL*Plus.

4.2.3 Running SQL Commands

The SQL command language enables you to manipulate data in the database.

Example: 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.

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. SQL*Plus will display 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. SQL*Plus processes the command and displays the results on the screen:

```
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 displays the command prompt again. If you made a mistake and therefore did not get the results shown above, simply re-enter the command. The headings may be repeated in your output, depending on the setting of a system variable called PAGESIZE. Sometimes, the result from a query will not fit the available page

width. You will need to adjust a system variable called LINESIZE, which sets the width of the output in characters. Typically, in the examples in this guide this is set to 70 characters. You may need to SET LINESIZE to 70 so the query output appears the same as in this guide. Whether you see the message concerning the number of records retrieved depends on the setting of a system variable called FEEDBACK. You will learn more about system variables in "System Variables that Affect How Commands Run". To save space, the number of records selected will not be shown in the rest of the examples in this guide.

4.3 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.

4.3.1 Dividing an SQL Command into Separate Lines

You can divide your SQL command into separate lines at any points you wish, as long as individual words are not split between lines. Thus, you can enter the query.

Example: "Entering a SQL Command" 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:*** "Entering a SQL Command", 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.

4.3.2 Ending an SQL Command

You can end an 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 : "Entering a SQL Command", and press Return. SQL*Plus will process the command and store it in the SQL buffer. 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 to run the command.

NOTE: You cannot enter a comment on the same line after a semicolon.

A slash (/) on a line by itself also tells SQL*Plus that you wish to run the command. Press Return at the end of the last line of the command. SQL*Plus prompts you with another line number. Type a slash and press Return again. SQL*Plus 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.

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. If you subsequently enter another SQL command, SQL*Plus overwrites the previous command in the buffer.

4.3.3 Creating Stored Procedures

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

CREATE FUNCTION

CREATE 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. 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

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

EXECUTE :ID := EMPLOYEE_MANAGEMENT.GET_ID('BLAKE')

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

4.3.4 Running PL/SQL Blocks

You can also use PL/SQL subprograms (called blocks) to manipulate data in the database. To enter a PL/SQL subprogram in SQL*Plus, you need to be in PL/SQL mode. You are placed in PL/SQL

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

You type a SQL command (such as CREATE FUNCTION) that creates a stored procedure. After you enter PL/SQL mode in this way, type the stored procedure you want to create. SQL*Plus treats PL/SQL subprograms in the same manner as SQL commands, except that a semicolon (;) or a blank line does not terminate and execute a block. Terminate PL/SQL subprograms by entering a period (.) by itself on a new line. You can also terminate and execute a PL/SQL subprogram by entering a slash (/) by itself on a new line.

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

SQL*Plus sends the complete PL/SQL subprogram to Oracle for processing (as it does SQL commands). 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;
  .
/
```

When you run a subprogram, the SQL commands within the subprogram may behave somewhat differently than they would outside the subprogram.

4.4 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 to one or a few letters.

Example: 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. On the command-line, 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.

Enter the RUN command to re-run the most recent query

RUN

```
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.
```

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

4.4.1 Understanding SQL*Plus Command Syntax

SQL*Plus commands have a different syntax from SQL commands or PL/SQL blocks. Continuing a Long SQL*Plus Command on Additional Lines you can continue a long SQL*Plus command by typing a hyphen at the end of the line and pressing Return. If you wish, you can type a space before typing the hyphen. SQL*Plus displays a right angle-bracket (>) as a prompt for each additional line.

For example:

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.

4.4.2 Ending a SQL*Plus Command

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

4.4.3 System Variables that affect How Commands Run

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

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

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

4.4.4 Saving Changes to the Database Automatically

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

Example: 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:

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

4.4.5 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 and returns to the command prompt.

In iSQL*Plus, click the Cancel button.

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.

4.4.6 Running Host Operating System Commands

You can execute a host operating system command from the SQL*Plus command prompt. This is useful when you want to perform a task such as listing existing host operating system files. To run a host operating system command, enter the SQL*Plus command HOST followed by the host operating system command. For example, this SQL*Plus command runs a host command, DIRECTORY *.SQL:

HOST DIRECTORY *.SQL

When the host 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 effect the current SQL*Plus session. For example, setting an operating system environment variable does not effect the current SQL*Plus session, but may effect SQL*Plus sessions started subsequently.

You can suppress access to the HOST command.

4.4.7 Getting Help

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

4.4.8 Listing a Table Definition

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

Example: Using the DESCRIBE Command

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

```
DESCRIBE EMP_DETAILS_VIEW;
 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 data dictionary. You can also use SQL SELECT commands to access this and other information in the database.

4.5 SQL*PLUS FUNCTIONS

SQL*Plus provides specialized functions to perform operations using the Data Manipulation Commands. A SQL function is a routine that performs a specific operation and returns the result. It is similar to a procedure, except that a procedure does not return a value. A function can take one or more arguments. One can broadly classify functions into single row functions and group functions.

Single Row Functions

A single row function or a scalar function returns only one value for every row required in the table. Single row function can appear in a select command and also be included in a 'where' clause. The single row function can be broadly classified as:

- Date functions
- Numeric functions
- Character functions
- Conversion functions
- Miscellaneous functions

Date Functions: They operate on date values producing output, which also belongs to date datatype, except for months_between date functions, which returns a number. We shall discuss some of the most important date functions with examples.

- Add_months

The add_month date function returns a date after adding a specified date with the specified number of months. The format is **add_months (d, n),** where d is the date and n represents the number of months.

Consider the following example to understand the above concept:

Example: SQL > select del_date, add_months (del_date, 2) from oreder_master;

The result will be

```
DEL_DATE                ADD_MONTH
----------              ----------
06-jan-88               06-mar-88
25-may-88               25-july-88
05-feb-88               05-apr-88
30-jun-88               30-aug-88
27-aug-88               27-oct-88
```

- Last_day

The format is last_day (d), which returns the date corresponding to the last day of the month.

Example: `SQL > select sysdate, last_day (del_date_) from order_master where odate>' 01-dec-88;`

The output for the above query will be

```
SYSDATE         LAST_DAY
--------        ---------
27-jan-88             31-jan-88
27-feb-88             28-feb-88
```

The above command will display the system date and the last day of the month to which del_date belongs.

NOTE: The sysdate variable will display system date (in words, current date).

- Months_between

To find out the number of months between two dates, we use the months_between function. Its format is

Months_between (d1, d2)

Where d1, d2 are dates. The output will be a number. If d1 is later than d2, result is positive; if earlier, negative. If d1 and d2 are either the same days of the month or both last days of the months, the result is always an integer; otherwise Oracle calculates the fractional portion of the result based on a 31-day month and considers the difference in time components of d1 and d2.

To find out the time between the odates and del_dates so that a schedule can be prepared to complete the orders before the delivery dates, the query in example below is used.

Example: `SQL > select months_between (del_date, odate) from order_master;`

The above example displays the number of months between the two dates

- Round

This function returns the date, which is rounded to the unit specified by the format model. Its format is

Round (d, [fmt])

Where d is date and fmt is the format model. Fmt is only an option; by default date will be rounded to the nearest day.

Example: `SQL > select del_date, round (del_date,'year') from order_master where vencode='v001;`

This request results in

DEL_DATE	ROUND (DEL
25-may-88	01-jan-88

Since format specified is ' year ' the argument, del_dates' value, is rounded to the nearest year. If the del_date was greater than 01-jun-88 then it would be rounded to the following year i.e. 89.

- Next_day

The format for the function is

next_day (d, day)

Where d represents date and day implies any weekday. This function can be illustrated with the following example

Example: `SQL > select next_day (sysdate,' Tuesday ') from dual;`

The Tuesday that immediately follows the sysdate will be displayed.

NOTE: dual is a system table. It is a table, which is automatically created by Oracle with the data dictionary. Dual table has one column defined to be of varchar2 datatype and contains only one row with value 'x'.

- Truncate

Truncate function returns the date with the time portion of the day truncated to the unit specified by the format model. The syntax is

truncate (d, [fmt])

if fmt is neglected, then date is converted to the nearest day.

Example: `SQL > select truncate (sysdate, 'year') from dual;`

If sysdate is '27-jan-88' the truncated result will be ' 01-jan-88 '.

Example: `SQL > select truncate (sysdate, ' month ') from dual;`

'01-Feb-88' will result from this query for the sysdate which is ' 27 -Jan -88'

Example: `SQL > select truncate (sysdate, ' day') from dual;`

'24-jan-99' will be the result because it rounds '27-jan-99' to the nearest Sunday.

Example: `SQL > select truncate (sysdate) from dual;`

The above statement does not include fmt, and therefore it is rounded to nearest day i.e. the sysdate.

- Greatest

The function is greatest (d1, d2 . . .), where d1, and d2 are dates. This function returns the latest date present in the argument.

Consider the following example, which will display the later date in the list.

The query in example below can be used to verify if the delivery dates for orderno 'o001' have been surpassed.

Example:
```
SQL > select del_date, sysdate, greatest (del_date, sysdate) from order_master
where orderno='o001';
```

DEL_DATE	SYSDATE	GREATEST
06-jan-99	27-jan-99	27-jan-99
05-feb-99	27-jan-99	05-feb-99

- New_time

The new_time function displays the time and date of date column or literal date in other time zones.

The format is displayed below

new_time (date, 'this', 'other');

This is replaced by a three-letter abbreviation of the current time zone while 'other' is replaced by a three-letter abbreviation of the time zone in which date is wanted.

Example: `SQL > select new_time ('13-feb-99', 'est.', 'yst') from dual;`

It returns 12-feb-99, which is the date in the time zone 'yst'.

Time Zones are as follows:

AST/ADT	Atlantic standard/day light time
BST/BDT	Bering standard/day light time
CST/CDT	Central standard/day light time
EST/EDT	Eastern standard/day light time
GMT	Greenwich mean time
HST/HDT	Alaska-Hawaii standard/day light time
MST/MDT	Mountains standard/day light time
NST	Newfoundland standard time
PST/PDT	Pacific standard/day light time
YST/YDT	Yukon standard/day light time

Character Functions

Character functions accept character input and return either character or number values. The character functions supported by Oracle are listed below.

Function	Input	Output
Initcap (char)	Select initcap ('hello') from dual;	Hello
Lower (char)	Select lower ('FUN') from dual;	fun
Upper (char)	Select upper ('sun') from dual;	SUN
Ltrim (char, set)	Select ltrim ('xyzadams','xyz' from dual;	adams
Rtrim (char, set)	Select rtrim ('xyzadams', 'xyz') from dual;	xyzad
Translate (char, from, to)	Select translate ('jack', 'j', b') from dual;	back
Replace (chat, searchstring, [rep, string])	Select replace ('jack and jue', 'j', 'bl') from dual;	blue
Substr (char, m, n)	Select substr ('abcdefg', 3, 2) from dual;	cd

NOTE: soundex is also a character function compares words that are spelled differently, but sound alike. Consider the following example

Example: `SQL > select venname from vendor_master where soundex (venname) = soundex ('sumesh');`

It returns 'somesh', present in the vendor_master table unless it has already been update or deleted.

Character functions accept character input and returns either character or number values. The first among the character functions is '***chr***'. This returns the character value for the number given within braces.

Example: `SQL > select chr (67) from dual;`

The above statement returns the value 'C' which is the character equivalent of the number 67. The ***chr*** function returns the character equivalent to the number in the braces.

The next function is the '***lpad***'. This takes three arguments. The first argument is the character string, which has to be displayed with the left padding. The second is the number, which indicates the total length of the return value. The third is the string, with which the left padding has to be done when required. An example gives a better understanding of the concept.

Example: `SQL > select lpad ('function', 15, '=') from dual;`

The output gives the sign '=' before the word function

LPAD ('FUNCTION'

=======function

The entire string is 15 in length after the padding is done.

The rpad function does the exact opposite of the lpad function. The number of arguments it takes is the same as the lpad function.

Example: `SQL > select rpad ('function', 15, '=');`

The rpad function pads the value to the right of the given string and is displayed as given below.

RPAD ('FUNCTION'

---

function=======

- Trim Function

This combines the functionality of the Ltrim and Rtrim. When specified leading, the function is similar to the Ltrim function and Oracle removes any leading characters equal to trim_character.

Example: `SQL > select trim (leading 9 from 99998769789999) from dual;`

The output trims off the all the 9's from beginning of the string. As soon as it encounters a character other than 9, it stops its action.

TRIM (LEAD

8769789999

When specified trailing, the function is similar to the Rtrim function and Oracle removes any trailing characters equal trim_character.

Example: `SQL > select trim (trailing 9 from 99998769789999) from dual;`

TRIM (TRAI

9999876978

- Length

When the length function is used in a query it returns the length of the string.

Example: `SQL > select length ('rohit') from dual;`

The output is 5.

- DECODE

Unlike the translate function which performs a character by character replacement the DECODE function does a value by value replacement

Select decode (<value, if1, then1 if2, then2, ...>) from <table_name>;

Example: `SQL > select vencode, decode (venname, 'rohit', 'rahul') name, tel_no from vendor_master where vencode='v001';`

The output is:

```
VENCO    NAME          TEL_NO
----     ----          -------
v001     rahul         1234567
```

- Concatenation (| |) Operator

The concatenation operator is used to merge two or more strings, or a string and a data value together.

Example: `SQL > select (' The address of '| | venname| |' is '| | venadd | |' '| |tel_no) address from vendor_master where vencode='v001';`

The output of this select statement is :

ADDRESS

The address of rohit is home 1234567

Numeric Functions

Numeric functions accept numeric input and return numeric value as the output. The values that the numeric functions return are accurate up to 38 decimal digits. The following tabular column will give you brief idea of the numeric functions supported by Oracle.

In addition to the functions that are already present a few new functions have been introduced. One among them is the ***In.*** this function returns the logarithmic value of the given number.

FUNCTION	INPUT	OUTPUT
Abs	Select abs (-15) from dual;	15
Ceil (n)	Select ceil (44.778) from dual;	45
Cos (n)	Select cos (180) from dual;	-.5984601
Cosh (n)	Select cosh (0) from dual;	1
Exp (n)	Select exp (4) from dual;	54.59815
Floor (n)	Select floor (100.2) from dual;	100
Power (m, n)	Select power (4,2) from dual;	16
Mod (m, n)	Select mod (10, 3) from dual;	1
Round (m, n)	Select round (100.256, 2) from dual	100.26
Trunc (m, n)	Select trunc (100.256, 2) from dual;	100.25
Squrt (n)	Select squrt (4) from dual;	2

Example: `SQL > select In (2) from dual;`

The output of the select statement is

```
LN (2)
———-
.69314718
```

Conversion Functions

Conversion function converts a value from one datatype to another. The conversion functions are broadly classified into the following:

To_char () transform DATE and NUMBER into character string

To_date () transform NUMBER, CHAR or VARCHAR2 into a DATE

To_number () transform CHAR or VARCHAR2 into a NUMBER

Why is this information important? To_Date is obviously necessary to accomplish date arithmetic. To_Char allows you to manipulate number as if it were a string, using string functions. To_Number allows you to use a string happens to contain only numbers as if it were a number; by using it you van add divide, subtract and so on.

- o_char ( )

The function is to_char (d, [fmt]), where d is date; fmt is the format model, which specifies the format of date. To_char conversion function concerts date to a value of varchar2 type in a form specified by the format fmt. If fmt is neglected then it converts date to varchar2 in the default date format. Consider the following example.

Example: `SQL > select to_char (sysdate, 'ddth "of" fmmonth yyyy') from dual;`

The above statement displays the date according to the format specified in the format model.

TO_CHAR (SYSDATE,'DDTH"

18th of august 2001

- To_date ( )

The format is to_date (char, [fmt]). This converts char or varchar datatype to date datatype. Format model, fmt specifies the frmt of character. Consider the following example, which returns date for the string 'January 27 1999'.

Example: `SQL > select to_date ('January 27 1999', 'month-dd-yyyy') from dual;`

The statement displays the following result.

TO_DATE ('

27-JAN-99

NOTE: The definition for fmt, explained in the to_char conversion function also holds good for to_date conversion function. The square braces indicate that the arguments are optional. The to_date

function can also be used in association with date functions. One instance where this is useful is explained below.

Example: `SQL > select round (to_date ('27-Jan- 1999'), 'year') from dual;`

The following result is displayed after execution of the above command

```
ROUND (TO_
------------
01-JAN-99
```

- To_number ( )

The to_number function allows the conversion of string concatenating numbers into the number datatype on which arithmetic operations can be performed. His is largely unnecessary as Oracle does an implicit conversion of numbers contained in a string.

Example: `SQL > select to_number ('100) from dual;`

The output is:

```
TO_NUMBER ('100')
-----------------
              100
```

Miscellaneous Functions

The following are some of the miscellaneous functions supported by Oracle.

- Uid
- User
- Nvl
- Vsize

Let us consider the functions mentioned above by one in dual.

- Uid

This function returns the integer value corresponding to the user currently logged in. the following example is illustrative.

Example: `SQL > select uid from dual;`

The result could be a number.

- USER

This function returns the login's user name, which is in varchar2 datatype. Consider the following example.

Example: `SQL > select user from dual;`

The result will be name of the current user.

```
USER
-----
SCOTT
```

- Null Value (nvl)

The Null value function is used in case where we want to consider Null values as zeros. The syntax is given as nvl (expression1, expression2).

If expression1 is Null, nvl will return expression2.

If expression1 is Not Null, nvl will return expression1.

If expression1 and expression2 are of different datatypes, then Oracle converts expression 2 to the datatype of expression1 and then compares it.

At Tom Dick and Harry Spares Inc a check is being made on whether the itemrate has been left out for items specified in the 'where' condition.

Example: `SQL > select itemrate, nvl (itemrate,0) from itemfile where itemcode ='i201' or itemdesc='bolts';`

The query will return the following output.

```
ITEMDESC            NUL (ITEMRATE, 0)
--------            -----------------
nuts                              0
bolts                          16.5
```

NOTE: Null values and zeros are not equivalent. Null values are represented by blank and zeros are represented by (0).

- Vsize

The function is vsize (expr). It returns the number of bytes in the expression. If expression is Null, it returns Null.

Example: `SQL > select vsize ('hello') from dual;`

The output is as follows

```
VSIZE ('HELLO')
---------------
              5
```

Group Functions

A group function processes a group of rows in the table, and return the result. Most of them act on all the rows of the default table. They accept the following parameters:

- DISTINCT

Makes the function act only on the rows that have different values.

- ALL

Makes the function consider all the values (rows), including duplicates. This is the default. With the exception of the COUNT function, all other functions ignore null values.

The group functions supported by Oracle are summarized below:

NOTE: In the following examples used to explain GROUP FUNCTIONS a in built table of Oracle emp will be used.

- AVG

The avg function will return the values of the column specified in the argument of the column.

Example: `SQL > select avg (sal) from emp;`

The query results in the average salary in the emp table under sal column.

- Min Function

This function will give the least of all the values present in the column.

Example: `SQL > select min (sal) from emp;`

The query results in the minimum salary present in the emp table.

- Max Function

This function returns the maximum value present in the column.

Example: `SQL > select max (sal) from emp;`

The output of this query depends on the EMP table generated by Oracle as default

```
MAX(SAL)
--------
    5000
```

- Count

Returns the number of count of rows of the query.

Example: `SQL > select count (sal) from emp;`

The output is shown below:

```
COUNT (SAL)
-----------
         15
```

Sum Function

The sum function can be used to obtain the sum of a range of values of a record set.

Example: `SQL > select sum (sal) from emp;`

The result obtained is

```
SUM (SAL)
--------
    34025
```

Up to this point you have seen that SQL can **select** rows of information from database tables, how the **where** clause can limit number of rows only those that meet certain rules that you define and how the rows returned can be sorted in ascending or descending sequence using **order by** clause.

You have also seen how the values in column can be modified by CHARACTER, NUMBER and DATE functions, and how groups can tell you about whole set of rows. Beyond the group functions you have seen there are also two group clauses, **having and group by.** These are parallel to where and order by clauses except that they act on groups and not on individual rows. These clauses can provide very powerful insights to your data.

4.5.1 Group By Clause

The following example lists the max commission based on unique salary in emp table.

Example:
```
SQL> select sal, max (comm) from emp group by sal;

       SAL MAX   (COMM)
---------   -----
       800
       950
      1100
      1250       1400
      1300
      1500          0
      1600        300
      2450
      2850
      2975
      3000
      5000       2345

12 rows selected.
```

The records in emp table of Oracle are as follows.

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.

Having Clause

The following example gives the clear view of having clause

SQL > select sal, max (comm) from emp group by sal having 5000 not in sal;

The output will be the following

```
       SAL MAX          (COMM)
---------          -------
       800
       950
      1100
      1250             1400
      1300
      1500                0
      1600              300
      2450
      2850
      2975
      3000
11 rows selected.
```

In the above example those unique salaries are selected based on the max commission received where salary does not include Rs. 5000.

Order of Execution of Various Clauses

1. Choose those rows based on where clause.
2. Group those rows together based on group by clause.
3. Calculate the results functions for each group.
4. Choose and eliminate group based on having clause.
5. Order the groups based on results of the group function in the order by clause, the order by clause must use either a group function or a column specified in a group by clause.

The order of execution is important because it has direct impact on the performance of your queries. In general, the more records that can be eliminated via where clause, the faster the query will execute. This performance benefits due to the reduction in numbers of rows that must be processed during the group by operation.

Check Your Progress

Fill in the blanks:

1. SQL*Plus processes the command and re-displays the ..... when ready for another command.
2. A ..... tells SQL*Plus that you want to run the command.
3. SQL*Plus ..... the subprograms you enter at the SQL*Plus command prompt in the SQL buffer.
4. Conversion function converts a ..... from one datatype to another.

4.6 LET US SUM UP

The area where SQL*Plus stores your most recently entered SQL command or PL/SQL block (but not SQL*Plus commands) is called the SQL buffer. The command or block remains there until you enter another. If you want to edit or re-run the current SQL command or PL/SQL block, you may do so without re-entering it. 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. You can also use PL/SQL subprograms (called blocks) to manipulate data in the database. To enter a PL/SQL subprogram in SQL*Plus, you need to be in PL/SQL mode. You are placed in PL/SQL mode when you type DECLARE or BEGIN at the SQL*Plus command prompt. After you enter PL/SQL mode in this way, type the remainder of your PL/SQL subprogram. SQL*Plus provides specialized functions to perform operations using the Data Manipulation Commands. A SQL function is a routine that performs a specific operation and returns the result. It is similar to a procedure, except that a procedure does not return a value.

4.7 KEYWORDS

Date Functions: They operate on date values producing output, which also belongs to date datatype, except for months_between date functions, which returns a number.

Sum Function: The sum function can be used to obtain the sum of a range of values of a record set.

Conversion Functions: Conversion function converts a value from one datatype to another.

To_number ( ): The to_number function allows the conversion of string concatenating numbers into the number datatype on which arithmetic operations can be performed.

4.8 QUESTIONS FOR DISCUSSION

1. Give an example to run a SQL command.
2. What is the processor to create a stored procedure?
3. Explain the SQL*Plus Command Syntax.
4. How the changes can be saved to the Database Automatically?
5. Discuss the four forms of Date functions.
6. What is the difference between the numeric functions and conversion functions?

Check Your Progress: Model Answers

1. command prompt
2. semicolon (;)
3. stores
4. value

4.9 SUGGESTED READINGS

Peter Rob, Carlos Coronel, *Database Systems: Design, Implementation and Management*, Seventh edition, Thomson Learning, 2007

Silberschatz, Korth, Sudarshan, *Database System Concepts*, Fifth edition, McGraw-Hill, 2005

Elmasari Navathe, *Fundamentals of Database Systems*, Third edition, Pearson Education Asia, 2001

E. J. Yannakoudakis, *The Architectural Logic of Database Systems*, Springer-Verlag, Digitized 2007

Fred R. McFadden, Jeffrey A. Hoffer, *Database Management*, Benjamin/Cummings, Digitized 2007

Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems*, Third edition, McGraw-Hill Higher Education, 2003

UNIT III

LESSON

5

SCHEMA OBJECTS

CONTENTS

5.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of schema objects
- Discuss data integrity

- Create and maintain tables
- Identify and explain the indexes sequences views
- Discuss the various user privileges and roles
- Explain the concept of synonyms

5.1 INTRODUCTION

Schema objects are logical structures created by users to contain, or reference, their data. Schema objects contain structures like tables, views, and indexes. You can create and manipulate schema objects using Oracle Enterprise Manager.

5.2 SCHEMA OBJECTS

A **schema** is a collection of logical structures of data, or schema objects. A schema is possessed by a database user and has the same name as that user. Each user owns a single schema. Schema objects can be created and manipulated with SQL and include the following types of objects:

- Clusters
- Database links
- Database triggers
- Dimensions
- External procedure libraries
- Indexes and index types
- Java classes, Java resources, and Java sources
- Materialized views and materialized view logs
- Object tables, object types, and object views
- Operators
- Sequences
- Stored functions, procedures, and packages
- Synonyms
- Tables and index-organized tables
- Views

Further types of objects are also stored in the database and can be created and manipulated with SQL but are not contained in a schema:

- Contexts
- Directories
- Profiles

- Roles
- Tablespaces
- Users

Schema objects are logical data storage structures. Schema objects do not have a one-to-one correspondence to physical files on disk that store their information. Though, Oracle stores a schema object logically within a tablespace of the database. The data of each object is physically contained in one or more of the tablespace's datafiles. For some objects, such as tables, indexes, and clusters, you can identify how much disk space Oracle allocates for the object within the tablespace's datafiles.

There is no association between schemas and tablespaces: a tablespace can contain objects from different schemas, and the objects for a schema can be contained in different tablespaces.

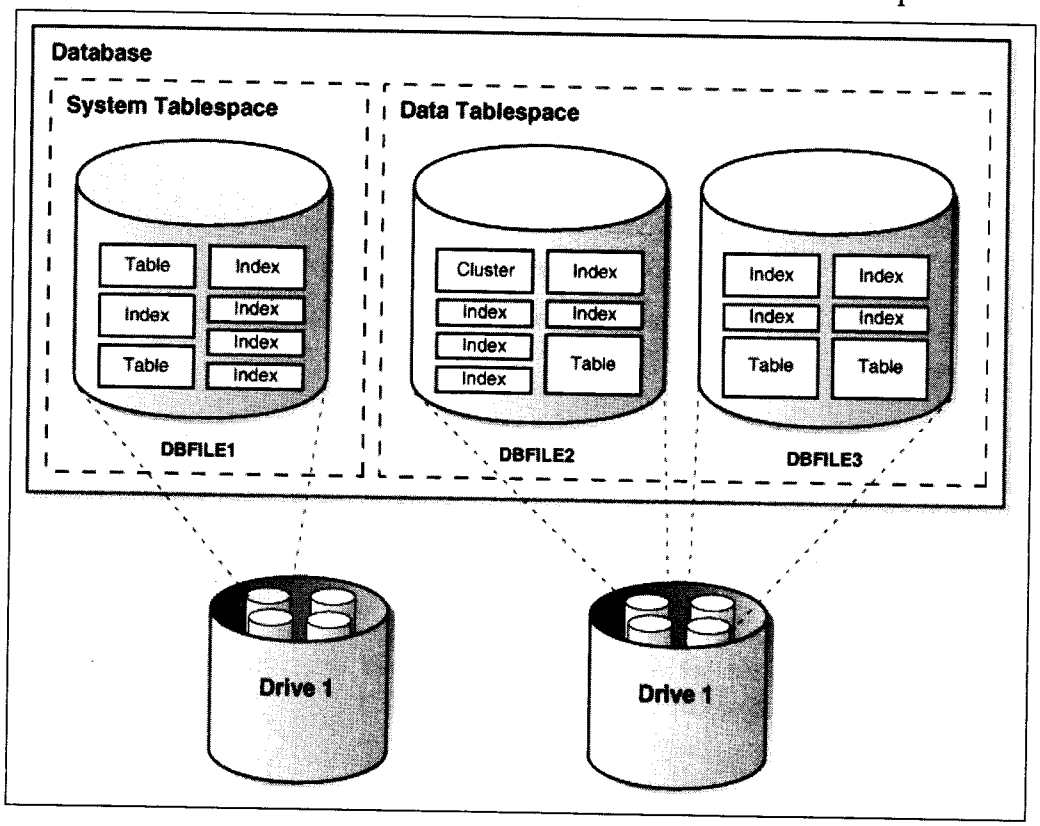


Figure 5.1: Schema Objects, Tablespaces, and Datafiles

5.3 DATA INTEGRITY

It is important that data remain to a predefined set of rules, as determined by the database administrator or application developer. As an example of data integrity, consider the tables EMP and DEPT and the business rules for the information in each of the tables, as illustrated in Figure 5.2.

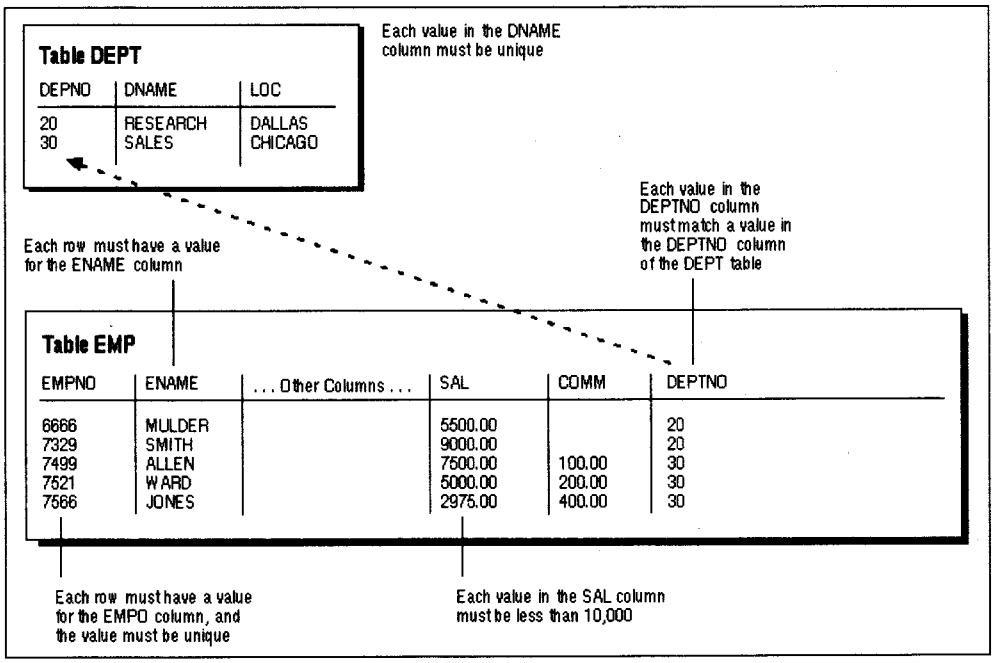


Figure 5.2: Examples of Data Integrity

Note that certain columns of each table have specific rules that constrain the data contained within them.

5.3.1 Types of Data Integrity

The following types of rules are applied to tables and enable you to enforce diverse types of data integrity.

Nulls

A rule defined on a single column that allows or disallows inserts or updates of rows containing a null for the column.

Unique Column Values

A rule defined on a column (or set of columns) that allows only the insert or update of a row containing a unique value for the column (or set of columns).

Primary Key Values

A rule defined on a column (or set of columns) so that each row in the table can be exclusively identified by the values in the column (or set of columns).

Referential Integrity

A rule defined on a column (or set of columns) in one table that permit the insert or update of a row only if the value for the column or set of columns (the dependent value) matches a value in a column of a related table (the referenced value).

Referential integrity also contain the rules that dictate what types of data manipulation are allowed on referenced values and how these actions affect dependent values. The rules associated with referential integrity include:

- ***Restrict:*** A referential integrity rule that disallows the update or deletion of referenced data.
- ***Set to Null:*** When referenced data is updated or deleted, all associated dependent data is set to NULL.

- ***Set to Default:*** When referenced data is updated or deleted, all associated dependent data is set to a default value.
- ***Cascade:*** When referenced data is updated, all associated dependent data is equally updated; when a referenced row is deleted, all associated dependent rows are deleted.

Complex Integrity Checking

A user-defined rule for a column (or set of columns) that allows or disallows inserts, updates, or deletes of a row based on the value it include for the column (or set of columns).

5.4 CREATING AND MAINTAINING TABLES

Tables are the essential unit of data storage in an Oracle database. Data is stored in **rows** and **columns**. You define a table with a **table name** (such as employees) and set of columns. You provide each column a **column name** (such as employee_id, last_name, and job_id), a **datatype** (such as VARCHAR2, DATE, or NUMBER), and a **width**. The width can be prearranged by the datatype, as in DATE. If columns are of the NUMBER datatype, define **precision** and **scale** instead of width. A row is a collection of column information corresponding to a single record.

You can state rules for each column of a table. These rules are called **integrity constraints**. One example is a NOT NULL integrity constraint. This constraint forces the column to contain a value in every row.

After you create a table, insert rows of data using SQL statements. Table data can then be queried, deleted, or updated using SQL.

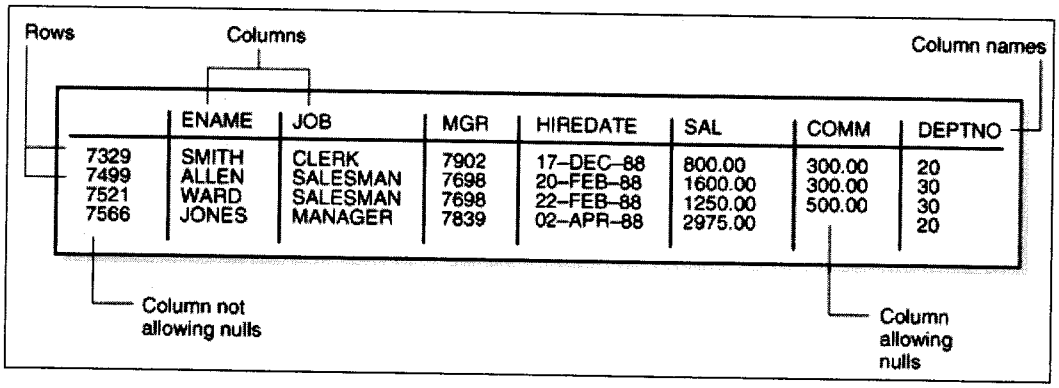


Figure 5.3: The EMP Table

5.4.1 How Table Data is Stored

When you create a table, Oracle repeatedly allocates a data segment in a tablespace to hold the table's future data. You can control the allocation and use of space for a table's data segment in the following ways:

- You can control the quantity of space allocated to the data segment by setting the storage parameters for the data segment.
- You can control the use of the free space in the data blocks that constitute the data segment's extents by setting the PCTFREE and PCTUSED parameters for the data segment.

Oracle stores data for a clustered table in the data segment formed for the cluster instead of in a data segment in a tablespace. Storage parameters cannot be specified when a clustered table is created or altered. The storage parameters set for the cluster always control the storage of all tables in the cluster.

A table's data segment (or cluster data segment, when dealing with a clustered table) is created in also the table owner's default tablespace or in a tablespace specifically named in the CREATE TABLE statement.

Row Format and Size

Oracle stores each row of a database table including data for less than 256 columns as one or more row pieces. If a whole row can be inserted into a single data block, then Oracle stores the row as one row piece. However, if all of a row's data cannot be inserted into a single data block or if an update to an existing row causes the row to outgrow its data block, then Oracle stores the row using multiple row pieces. A data block generally contains only one row piece for each row. When Oracle must store a row in more than one row piece, it is chained across multiple blocks.

When a table has more than 255 columns, rows that have data after the 255th column are possible to be chained within the same block. This is called intra-block chaining. A chained row's pieces are chained together using the rowids of the pieces. With intra-block chaining, users obtain all the data in the same block. If the row fits in the block, users do not see an effect in I/O performance, because no extra I/O operation is required to retrieve the rest of the row.

Each row piece, chained or unchained, includes a row header and data for all or some of the row's columns. Individual columns can also span row pieces and, consequently, data blocks. Figure 5.4 shows the format of a row piece:

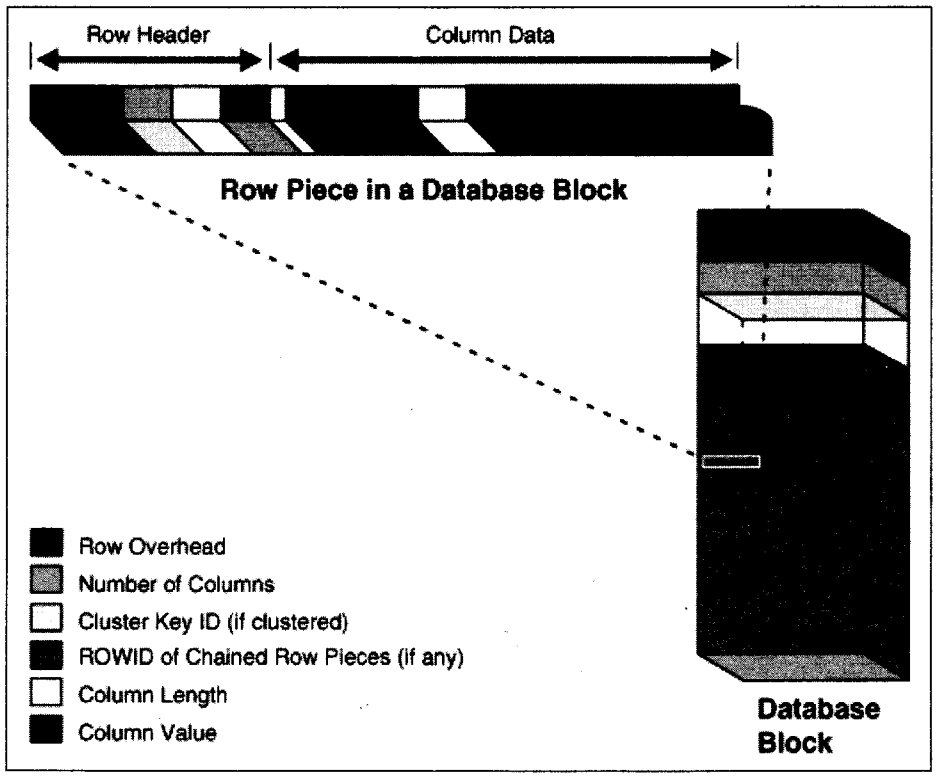


Figure 5.4: The Format of a Row Piece

The **row header** precedes the data and contains information about:

- Row pieces
- Chaining (for chained row pieces only)

- Columns in the row piece
- Cluster keys (for clustered data only)

A row fully controlled in one block has at least 3 bytes of row header. After the row header information, each row contains column length and data. The column length necessitates 1 byte for columns that store 250 bytes or less, or 3 bytes for columns that store more than 250 bytes, and precedes the column data. Space required for column data depends on the datatype. If the datatype of a column is variable length, then the space required to hold a value can grow and shrink with updates to the data.

To preserve space, a null in a column only stores the column length (zero). Oracle does not store data for the null column. Also, for trailing null columns, Oracle does not even store the column length. Clustered rows contain the same information as nonclustered rows. In addition, they contain information that references the cluster key to which they belong.

Rowids of Row Pieces

The **rowid** recognize each row piece by its location or address. After they are assigned, a given row piece retains its rowid until the corresponding row is deleted or exported and imported using Oracle utilities. For clustered tables, if the cluster key values of a row change, then the row keeps the same rowid but also gets an additional pointer rowid for the new values.

Because rowids are constant for the lifetime of a row piece, it is useful to orientation rowids in SQL statements such as SELECT, UPDATE, and DELETE.

Column Order

The column order is the same for all rows in a given table. Columns are generally stored in the order in which they were listed in the CREATE TABLE statement, but this is not guaranteed. For example, if a table has a column of datatype LONG, then Oracle constantly stores this column last. Also, if a table is altered so that a new column is added, then the new column becomes the last column stored.

In common, try to place columns that frequently contain nulls last so that rows take less space. Note, though, that if the table you are creating includes a LONG column as well, then the benefits of placing frequently null columns last are lost.

5.4.2 Table Compression

Oracle's table compression feature compresses data by abolishing duplicate values in a database block. Compressed data stored in a database block (also known as disk page) is self-contained. That is, all the information needed to restructure the uncompressed data in a block is available within that block. Duplicate values in all the rows and columns in a block are stored once at the beginning of the block, in what is called a symbol table for that block. All occurrences of such values are replaced with a short reference to the symbol table.

With the exception of a symbol table at the beginning, compressed database blocks look very much like regular database blocks. All database features and functions that work on regular database blocks also work on compressed database blocks.

Database objects that can be compressed contain tables and materialized views. For partitioned tables, you can choose to compress some or all partitions. Compression attributes can be declared for a tablespace, a table, or a partition of a table. If declared at the tablespace level, then all tables formed in that tablespace are compressed by default. You can alter the compression attribute for a table

(or a partition or tablespace), and the change only applies to new data going into that table. As a result, a single table or partition may include some compressed blocks and some regular blocks. This guarantees that data size will not increase as a result of compression; in cases where compression could increase the size of a block, it is not applied to that block.

Using Table Compression

Compression occurs while data is being bulk inserted or bulk loaded. These operations include:

- Direct path SQL*Loader
- CREATE TABLE and AS SELECT statements
- Parallel INSERT (or serial INSERT with an APPEND hint) statements

Accessible data in the database can also be compressed by moving it into compressed form through ALTER TABLE and MOVE statements. This operation takes a restricted lock on the table, and therefore prevents any updates and loads until it completes. If this is not acceptable, then Oracle's online redefinition utility (DBMS_REDEFINITION PL/SQL package) can be used.

Data compression works for all data types apart from for all variants of LOBs and data types derived from LOBs, such as VARRAYs stored out of line or the XML data type stored in a CLOB.

Table compression is done as part of bulk loading data into the database. The transparency associated with compression is most visible at that time. This is the primary trade-off that needs to be taken into account when considering compression.

Compressed tables or partitions can be customized the same as other Oracle tables or partitions. For example, data can be modified using INSERT, UPATE, and DELETE statements. Though, data modified without using bulk insertion or bulk loading techniques is not compressed. Deleting compressed data is as fast as deleting uncompressed data. Inserting new data is also as fast, since data is not compressed in the case of conventional INSERT; it is compressed only doing bulk load. Updating compressed data can be slower in some cases. For these reasons, compression is more suitable for data warehousing applications than OLTP applications. Data should be organized such that read only or infrequently changing portions of the data (for example, historical data) is kept compressed.

Nulls Indicate Absence of Value

A **null** is the absence of a value in a column of a row. Nulls designate missing, unknown, or inapplicable data. A null should not be used to imply any other value, such as zero. A column permit nulls unless a NOT NULL or PRIMARY KEY integrity constraint has been defined for the column, in which case no row can be inserted without a value for that column.

Nulls are stored in the database if they fall between columns with data values. In these cases they entail 1 byte to store the length of the column (zero).

Trailing nulls in a row require no storage as a new row header signals that the remaining columns in the previous row are null. For instance, if the last three columns of a table are null, no information is stored for those columns. In tables with many columns, the columns more likely to contain nulls should be defined last to conserve disk space.

Most evaluation between nulls and other values are by definition neither true nor false, but unknown. To identify nulls in SQL, use the IS NULL predicate. Use the SQL function NVL to convert nulls to non-null values.

Nulls are not indexed, apart from when the cluster key column value is null or the index is a bitmap index.

Default Values for Columns

You can give a default value to a column of a table so that when a new row is inserted and a value for the column is omitted or keyword DEFAULT is supplied, a default value is supplied routinely. Default column values work as though an INSERT statement actually specifies the default value.

The datatype of the default literal or expression must match or be adaptable to the column datatype.

If a default value is not explicitly defined for a column, then the default for the column is implicitly set to NULL.

Default Value Insertion and Integrity Constraint Checking

Integrity constraint checking occurs after the row with a default value is introduced. For example, in Figure 5.5 a row is inserted into the emp table that does not include a value for the employee's department number. Because no value is supplied for the department number, Oracle inserts the deptno column's default value of 20. After inserting the default value, Oracle checks the FOREIGN KEY integrity constraint defined on the deptno column.

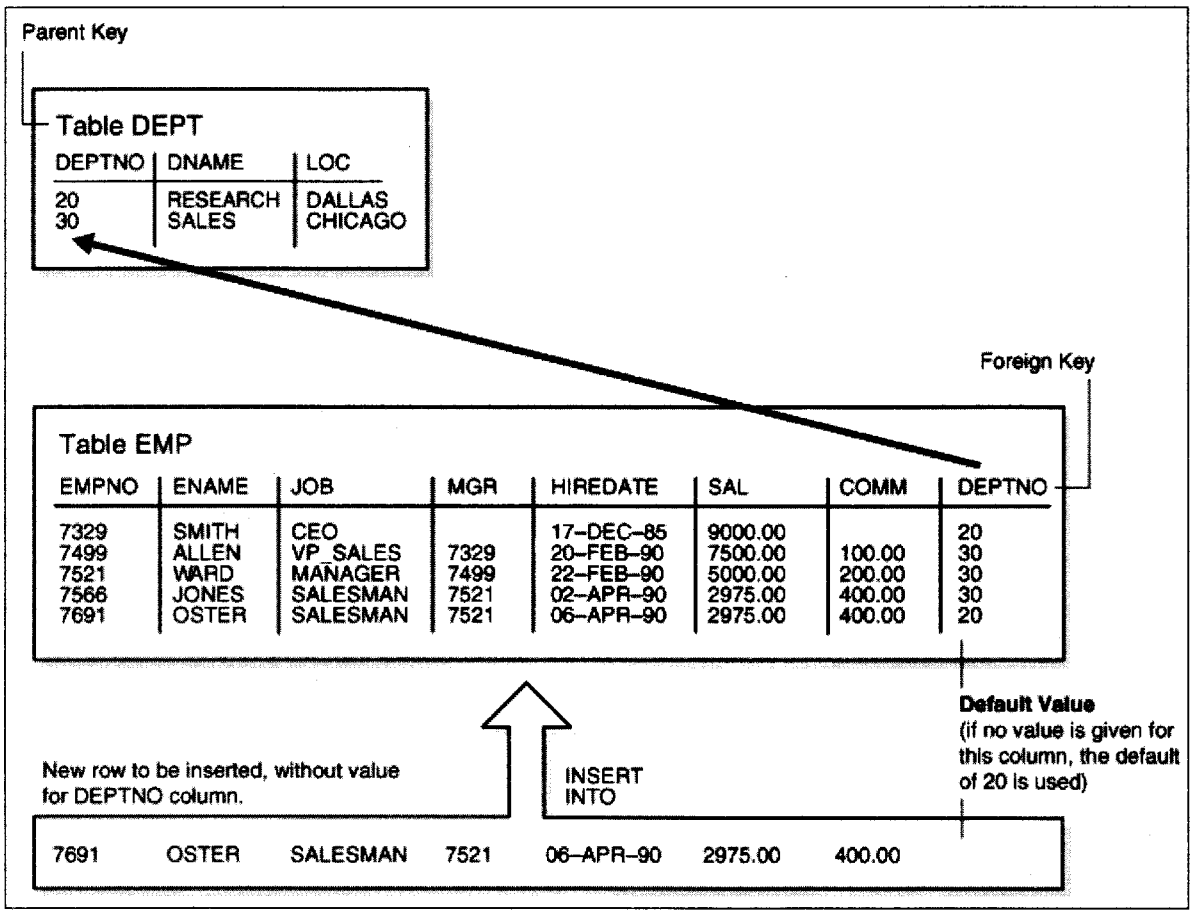


Figure 5.5: Default Column Values

5.4.3 Types of Tables

Partitioned Tables

Partitioned tables allow your data to be broken down into smaller, more controllable pieces called partitions, or even subpartitions. Indexes can be partitioned in similar fashion. Each partition can be managed independently, and can operate independently of the other partitions, thus providing a structure that can be better tuned for availability and performance.

Nested Tables

You can make a table with a column whose datatype is another table. That is, tables can be **nested** within other tables as values in a column. The Oracle database server stores nested table data out of line from the rows of the parent table, using a **store table** that is associated with the nested table column. The parent row contains a unique set identifier value associated with a nested table instance.

Temporary Tables

In addition to permanent tables, Oracle can make temporary tables to hold session-private data that subsist only for the duration of a transaction or session.

The CREATE GLOBAL TEMPORARY TABLE statement makes a temporary table that can be transaction-specific or session-specific. For transaction-specific temporary tables, data exists for the duration of the transaction. For session-specific temporary tables, data exists for the duration of the session. Data in a temporary table is private to the session. Each session can only see and modify its own data. DML locks are not acquired on the data of the temporary tables. The LOCK statement has no effect on a temporary table, because each session has its own private data.

A TRUNCATE statement subjected on a session-specific temporary table truncates data in its own session. It does not truncate the data of other sessions that are using the same table.

DML statements on temporary tables do not produce redo logs for the data changes. However, undo logs for the data and redo logs for the undo logs are generated. Data from the temporary table is routinely dropped in the case of session termination, either when the user logs off or when the session terminates abnormally such as during a session or instance failure.

You can generate indexes for temporary tables using the CREATE INDEX statement. Indexes created on temporary tables are also temporary, and the data in the index has the same session or transaction scope as the data in the temporary table.

You can create views that access together temporary and permanent tables. You can also create triggers on temporary tables.

Oracle utilities can export and import the meaning of a temporary table. However, no data rows are exported even if you use the ROWS clause. Similarly, you can replicate the definition of a temporary table, but you cannot replicate its data.

External Tables

External tables access data in external sources as if it were in a table in the database. You can connect to the database and generate metadata for the external table using DDL. The DDL for an external table consists of two parts: one part that describes the Oracle column types, and another part (the access parameters) that describes the mapping of the external data to the Oracle data columns.

An external table does not explain any data that is stored in the database. Nor does it describe how data is stored in the external source. Instead, it describes how the external table layer needs to present the data to the server. It is the responsibility of the access driver and the external table layer to do the necessary transformations required on the data in the datafile so that it matches the external table definition.

External tables are read only; consequently, no DML operations are possible, and no index can be created on them.

- The Access Driver

When you generate an external table, you specify its type. Each type of external table has its own access driver that provides access parameters unique to that type of external table. The access driver makes sure that data from the data source is processed so that it matches the definition of the external table.

In the framework of external tables, loading data refers to the act of reading data from an external table and loading it into a table in the database. Unloading data refers to the act of reading data from a table in the database and inserting it into an external table.

The default type for external tables is ORACLE_LOADER, which allow you read table data from an external table and load it into a database. Oracle also gives the ORACLE_DATAPUMP type, which lets you unload data (that is, read data from a table in the database and insert it into an external table) and then reload it into an Oracle database.

The definition of an external table is kept discretely from the description of the data in the data source. This means that:

- ❖ The source file can have more or fewer fields than there are columns in the external table
- ❖ The datatypes for fields in the data source can be dissimilar from the columns in the external table

- Data Loading with External Tables

The major use for external tables is to use them as a row source for loading data into an actual table in the database. After you create an external table, you know how to use a CREATE TABLE AS SELECT or INSERT INTO ... AS SELECT statement, using the external table as the source of the SELECT clause.

When you access the external table during a SQL statement, the fields of the external table can be used just like any other field in a regular table. In exacting, you can use the fields as arguments for any SQL built-in function, PL/SQL function, or Java function. This allows you manipulate data from the external source. For data warehousing, you can do more sophisticated transformations in this way than you can with simple datatype conversions. You can also use this mechanism in data warehousing to do data cleansing.

While external tables cannot contain a column object, constructor functions can be used to build a column object from attributes in the external table

- Parallel Access to External Tables

After the metadata for an external table is created, you can query the external data openly and in parallel, using SQL. As a result, the external table acts as a view, which lets you run any SQL query against external data without loading the external data into the database.

The degree of parallel access to an external table is particular using standard parallel hints and with the PARALLEL clause. Using parallelism on an external table allows for concurrent access to the datafiles that comprise an external table. Whether a single file is accessed concurrently is dependent upon the access driver implementation, and attributes of the datafile(s) being accessed (for example, record formats).

5.5 INDEXES SEQUENCES VIEWS

A sequence is a database object, which can generate unique, sequential integer values. Sequences help to ease the process of creating unique identifiers for a record in a database. A *sequence* is a simply an automatic counter, which is enabled whenever it is accessed. It can be used to automatically generate primary key or unique key values. A sequence can either be in ascending or descending order.

When a sequence is created, it adopts some default values that are adequate for most situations. A default sequence has the following characteristics:

- Always starts from number 1
- In ascending order
- Increases by 1

The syntax for creating a sequence is as follows:

CREATE SEQUENCE <name of sequence>

START WITH integer

INCREMENT BY integer

MINVALUE integer

NOMINVALUE

MAXVALUE integer

NOMAXVALUE

CYCLE

NOCYCLE

CACHE integer

NOCACHE sequence

ORDER

NOORDER

Here *START WITH* indicates the initial value of the sequence. When used for the first time, it returns the value specified by this clause. *INCREMENT BY* indicates the value by which the sequence will be incremented each time it is accessed. *MAXVALUE* indicates the maximum value that the sequence may have. When omitted, the maximum value of the sequence can be 1.00e + 27 or 10^{27}.

NOMAXVALUE indicates that the sequence doesn't have a predefined maximum value. *CACHE* indicates the number of sequences that should be created directly in the cache memory. In a situation

where there is a lot of access to the sequences, the higher the value specified, the less access to the disk will occur.

CYCLE/NOCYCLE:

Cycle indicates that the sequence should return to the initial value when the maximum value is reached. NOCYCLE, in turn prevents it from returning to the beginning.

After creating a sequence we can access the values with the help of pseudo columns like ***curval*** and ***nextval.*** Oracle has several pseudo - columns that behave as an extra column when a table is created.

curval and nextval

nextval returns initial value of the sequence, when referred to, for the first time. Later references to nextval will increment the sequence using the INCREMENT BY clause and return the new value.

curval returns the current value to the sequence which is the value returned by the last reference to the nextval.

Altering of Sequence

With the ALTER SEQUENCE command the user can change some of the sequence's parameters. The sequence can be altered when we want to perform the following:

- Set or eliminate minvalue or maxvalue
- Change the increment value
- Change the number of chaced sequence numbers

However there are some restrictions. You are not allowed, for example to change its initial value. The minimum value for the table cannot be greater than the current value.

5.5.1 Indexes

Indexes are optional structures associated with tables. We can create indexes explicitly to speed up SQL statement execution on a table. Similar to indexes in books that help us to locate information faster, an Oracle index provides a faster access to path to table data. The index points directly to the location of the rows containing the value. Indexes are the primary means of reducing disk I/O when properly used.

When to Create an Index

An index can be created during the design process of the table structure using the PRIMARY KEY constraint. However it is better to create it later, particularly when the has existing data that will be loaded with the utilities. In this case whenever a row is inserted, the index is updated, requiring more processing time.

Columns of type *lobs, long and long row* cannot be indexed.

We create an index on a column or combinations of columns using CREATE INDEX command as follows:

CREATE [UNIQUE] INDEX <name of index> ON <table name> (column name) ;

Example

Now we will create a index on DEPARTMENT table in login scott/tiger (username / password).

SQL > create index depname on department (name) ;

The output will be

Index created.

When we create an index, Oracle fetches and sort's columns to be indexed, and stores the ROWID along with the index value for each row. Then Oracle loads the index from the bottom up. Indexes are physically independent of the data in the associated table. We can create and drop an index at any time without effecting the base tables or other indexes. Indexes as independent structures require storage space.

5.5.2 Types of Index

Unique Indexes

Indexes can be unique or non-unique. Unique indexes guarantee that no two rows of a table have duplicate values in the columns that define the index. Non-unique indexes do not impose this restriction on the column values. Oracle enforces unique integrity constraints by automatically defining a unique index on the unique key. Using the CREATE UNIQUE INDEX command as follows creates a unique index, but this statement will fail if any duplicates already exist. If you use primary key constraint you will never have duplicates.

Syntax

SQL > CREATE UNIQUE INDEX <index name> on <table name (column name)> ;

If **create unique index** statement succeeds, then any future attempt to **insert** a row that would create a duplicate key will fail and result in this error message:

ERROR at line 1: ORA-00001 : unique constraint

(table name . column name) violated

NOTE: A unique index is automatically created when we create unique or primary key constraint. Alternatively a constraint is imposed on the column when we create a unique index. We cannot create index for a column which is already indexed.

Composite Index

A composite index (also called a concatenated index) is an index created on multiple columns of a table. Columns in a composite index can appear in any order and need not be adjacent columns of the table.

Composite index can enhance the speed of retrieving data for the select statement in which the 'where' clause references all or the leading portion of the columns in the composite index.

Reverse Key Index

Creating a reverse key index, when compared to a standard index, reverses each byte of the column being indexed while keeping the column order. Such an arrangement can help avoid performance degradation in indexes where modifications to the index are concentrated on a small set of blocks. By reversing the keys of the index, the insertions become distributed all over the index.

Creating a Bitmap Index

The advantages of using a bitmap indexes are greatest on the tables in which the data is infrequently updated, because they add to the cost of all data manipulation transactions against the tables they index.

Bitmap indexes are appropriate when nonselective columns are used as limiting conditions in a query. If you choose bitmap indexes, you will need to weight the performance benefit during queries against the performance cost during data manipulation commands. The more the bitmap indexes on the table, grater the cost will be on each transaction. You should bitmap index on those columns that frequently has new values added to it.

Advantages of using Bitmap Indexes

- Reduced response time for large classes of ad hoc queries.
- A substantial reduction of space usage as compared to other indexing techniques.
- Dramatic performance gains even on very low end hardware.

5.6 USER PRIVILEGES AND ROLES

A user **privilege** is a right to perform a particular type of SQL statement, or a right to access another user's object. The types of privileges are defined by Oracle.

Roles, on the other hand, are created by users (generally administrators) and are used to group together privileges or other roles. They are a means of facilitating the granting of numerous privileges or roles to users.

This section describes Oracle user privileges, and includes the following topics:

- System Privileges
- Object Privileges
- User Roles

5.6.1 System Privileges

There are over 100 separate system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations.

Restricting System Privileges

Because system privileges are so influential, Oracle recommends that you configure your database to prevent regular (non-DBA) users exercising ANY system privileges (such as UPDATE ANY TABLE) on the data dictionary. In order to protect the data dictionary, ensure that the O7_DICTIONARY_ACCESSIBILITY initialization parameter is set to FALSE. This feature is called the dictionary protection mechanism.

If you allow dictionary protection (O7_DICTIONARY_ACCESSIBILITY is FALSE), access to objects in the SYS schema (dictionary objects) is limited to users with the SYS schema. These users are SYS and those who connect as SYSDBA. System privileges providing access to objects in other schemas do *not* give other users access to objects in the SYS schema. For example, the SELECT ANY TABLE privilege let users to access views and tables in other schemas, but does not allow them to select dictionary objects (base tables of dynamic performance views, views, packages, and synonyms). These users can, however, be granted explicit object privileges to access objects in the SYS schema.

Accessing Objects in the SYS Schema

Users with explicit object privileges or those who connect with managerial privileges (SYSDBA) can access objects in the SYS schema. A different means of allowing access to objects in the SYS schema is by granting users any of the following roles:

- ***SELECT_CATALOG_ROLE:*** This role can be granted to users to allow SELECT privileges on all data dictionary views.
- ***EXECUTE_CATALOG_ROLE:*** This role can be granted to users to allow EXECUTE privileges for packages and procedures in the data dictionary.
- ***DELETE_CATALOG_ROLE:*** This role can be granted to users to allow them to delete records from the system audit table (AUD$).

As well, the following system privilege can be granted to users who require access to tables created in the SYS schema:

- SELECT ANY DICTIONARY

This system privilege permits query access to any object in the SYS schema, including tables created in that schema. It must be granted independently to each user requiring the privilege. It is not included in GRANT ALL PRIVILEGES, nor can it be granted through a role.

5.6.2 Object Privileges

Each type of object has different privileges associated with it.

You can specify ALL [PRIVILEGES] to grant or revoke all accessible object privileges for an object. ALL is not a privilege; rather, it is a shortcut, or a way of granting or revoking all object privileges with one word in GRANT and REVOKE statements. Note that if all object privileges are granted using the ALL shortcut, individual privileges can still be revoked.

Similarly, all individually granted privileges can be revoked by specifying ALL. Though, if you REVOKE ALL, and revoking causes integrity constraints to be deleted (because they depend on a REFERENCES privilege that you are revoking), you have to include the CASCADE CONSTRAINTS option in the REVOKE statement.

5.6.3 User Roles

A **role** groups more than a few privileges and roles, so that they can be granted to and revoked from users simultaneously. A role must be enabled for a user before it can be used by the user.

Oracle offer some predefined roles to help in database administration. You can grant privileges and roles to, and revoke privileges and roles from, these predefined roles in the same way as you do with any role you define.

Managing User Roles

This section describes aspects of managing roles, and contains the following topics:

- Creating a Role
- Specifying the Type of Role Authorization
- Dropping Roles

Creating a Role

You can create a role using the `CREATE ROLE` statement, but you must have the `CREATE ROLE` system privilege to do so. Usually, only security administrators have this system privilege.

You must give each role you create a unique name amongst existing usernames and role names of the database. Roles are not enclosed in the schema of any user. In a database that uses a multibyte character set, Oracle proposes that each role name contain at least one single-byte character. If a role name contains only multibyte characters, the encrypted role name/password combination is considerably less secure.

The following statement creates the clerk role, which is certified by the database using the password bicentennial:

CREATE ROLE clerk IDENTIFIED BY bicentennial;

The IDENTIFIED BY clause state how the user must be authorized before the role can be enabled for use by a specific user to which it has been granted. If this clause is not specified, or NOT IDENTIFIED is specified, then no authorization is required when the role is enabled. Roles can be specified to be authorized by:

- The database using a password
- An application using a specified package
- Externally by the operating system, network, or other external source
- Globally by an enterprise directory service

These authorizations are discussed in following sections.

Afterward, you can set or change the authorization method for a role using the `ALTER ROLE` statement. The following statement alters the `clerk` role to specify that the user must have been authorized by an external source before enabling the role:

ALTER ROLE clerk IDENTIFIED EXTERNALLY;

To alter the authorization method for a role, you must have the ALTER ANY ROLE system privilege or have been granted the role with the ADMIN OPTION.

Specifying the Type of Role Authorization

The methods of authorizing roles are obtainable in this section. A role must be enabled for you to use it.

- Role Authorization by the Database

The use of a role authorized by the database can be protected by an associated password. If you are arranged a role protected by a password, you can enable or disable the role by supplying the proper password for the role in a SET ROLE statement. Though, if the role is made a default role and enabled at connect time, the user is not required to enter a password.

The next statement creates a role manager. When it is enabled, the password morework must be supplied.

CREATE ROLE manager IDENTIFIED BY morework;

- Role Authorization by an Application

The INDENTIFIED USING *package_name* clause lets you make an application role, which is a role that can be allow only by applications using an authorized package. Application developers do not need to secure a role by embedding passwords inside applications. Instead, they can create an application role and identify which PL/SQL package is authorized to enable the role.

The following example designate that the role admin_role is an application role and the role can only be enabled by any module defined inside the PL/SQL package hr.admin.

CREATE ROLE admin_role IDENTIFIED USING hr.admin;

When enabling the user's default roles at login as specified in the user's profile, no checking is performed for application roles.

- Role Authorization by an External Source

The following statement creates a role named `accts_rec` and need that the user be authorized by an external source before it can be enabled:

CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;

- Role Authorization by the Operating System

Role authentication through the operating system is functional only when the operating system is able to dynamically link operating system privileges with applications. When a user starts an application, the operating system grants an operating system privilege to the user. The granted operating system privilege communicates to the role associated with the application. At this point, the application can enable the application role. When the application is terminated, the formerly granted operating system privilege is revoked from the user's operating system account.

If a role is authorized by the operating system, you have to configure information for each user at the operating system level. This operation is operating system dependent.

If roles are granted by the operating system, you do not require to have the operating system authorize them also; this is redundant.

- Role Authorization and Network Clients

If users connect to the database over Oracle Net, by default their roles cannot be authenticated by the operating system. This contains connections during a shared server configuration, as this connection requires Oracle Net. This restriction is the default because a remote user could impersonate another operating system user over a network connection.

If you are not worried with this security risk and want to use operating system role authentication for network clients, set the initialization parameter REMOTE_OS_ROLES in the database's initialization parameter file to TRUE. The change will take consequence the next time you start the instance and mount the database. The parameter is FALSE by default.

- Role Authorization by an Enterprise Directory Service

A role can be defined as a global role, whereby a (global) user can only be certified to use the role by an enterprise directory service. You define the global role nearby in the database by granting privileges and roles to it, but you cannot grant the global role itself to any user or other role in the database. When a global user attempts to connect to the database, the enterprise directory is queried to obtain any global roles associated with the user.

The following statement creates a global role:

CREATE ROLE supervisor IDENTIFIED GLOBALLY;

Global roles are one component of enterprise user organization. A global role only relates to one database, but it can be granted to an enterprise role defined in the enterprise directory. An enterprise role is a directory structure which have global roles on multiple databases, and which can be granted to enterprise users.

A common discussion of global authentication and authorization of users, and its role in enterprise user management, was presented earlier in "Global Authentication and Authorization".

Dropping Roles

In some cases, it may be appropriate to drop a role from the database. The security domains of all users and roles granted a dropped role are immediately changed to reflect the absence of the dropped role's privileges. All indirectly granted roles of the dropped role are also removed from affected security domains. Dropping a role automatically removes the role from all users' default role lists.

Because the creation of objects is not dependent on the privileges received through a role, tables and other objects are not dropped when a role is dropped.

You can drop a role using the SQL statement DROP ROLE. To drop a role, you must have the DROP ANY ROLE system privilege or have been granted the role with the ADMIN OPTION.

The following statement drops the role CLERK:

DROP ROLE clerk;

5.7 SYNONYMS

Synonyms are a simple way to access tables and other database objects using alternate name or a shortcut. A synonym is a database object, which is used as an alias (alternative name) for a table view or sequence. A reference is made to the original object when the synonym is created. For example, if a synonym is created for a table, Oracle associates the address of the table to the synonym, and does not create a duplicate of the table.

There are many advantages of using a synonym. They are discussed below :

- Simplify SQL statement
- Hide the name and owner of an object that is being specified
- Provide location transparency for remote objects of a distributed database. Provide public access to an object

If the object is changed or moved, all you have to do is to update the synonym, rather than change the numerous references to the object. Synonym helps the user in securing his objects and helps in simplifying execution of the commands.

A synonym can be public and visible to all users, or private and available only to user who created it. The private synonym is created by normal user, which is available to that person whereas the public synonym is created by the DBA, which can be availed by any database user. Synonyms can be used to replace objects in the following SQL commands.

- SELECT

- INSERT
- DELETE
- GRANT
- UPDATE
- REVOKE

The syntax for creating a synonym is analyzed as follows:

CREATE [PUBLIC] SYNONYM <name of synonym> for <table name > ;

Here PUBLIC creates a public synonym. If the PUBLIC clause is omitted, the synonym created will be private synonym belonging to the table (schema) or the user who created it.

The user must have the CREATE SYNONYM privilege to create a synonym. The vendor_master table has been created by marketing division and the user wants to grant all access on that table to accounts but he does not want him to alter the structure of the table or drop it all together. Consider the following example, which creates private synonym on the table vendor_master with a different name say, vmast.

Example

SQL > create synonym vmast for vendor_master ;

SQL > grant all on vmast to accounts ;

Now accounts can do all the DML manipulations such as insert, update, delete on the particular synonym.

5.7.1 Removing Synonym

To remove a synonym, the user must issue the DROP SYNONYM command.

Syntax

DROP [PUBLIC] SYNONYM <name of synonym> ;

All references made to a synonym that has been deleted generate an error. A synonym that is associated to a table is not automatically deleted with the table.

Check Your Progress

Fill in the blanks:

1. A schema is a collection of logical structures of ....., or schema objects.
2. When you create a table, Oracle automatically allocates a ..... in a tablespace to hold the table's future data.
3. Partitioned tables allow your data to be broken down into ..... , more manageable pieces called partitions,
4. Synonyms are a simple way to ..... tables and other database objects using alternate name or a shortcut.

5.8 LET US SUM UP

A **schema** is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. It is important that data adhere to a predefined set of rules, as determined by the database administrator or application developer. **Tables** are the basic unit of data storage in an Oracle database. Data is stored in **rows** and **columns**. You define a table with a **table name** (such as `employees`) and set of columns. A sequence is a database object, which can generate unique, sequential integer values. Sequences help to ease the process of creating unique identifiers for a record in a database. A user **privilege** is a right to execute a particular type of SQL statement, or a right to access another user's object. The types of privileges are defined by Oracle. Synonyms are a simple way to access tables and other database objects using alternate name or a shortcut. A synonym is a database object, which is used as an alias (alternative name) for a table view or sequence.

5.9 KEYWORDS

Schema: A schema is a collection of logical structures of data, or schema objects.

Tables: Tables are the basic unit of data storage in an Oracle database.

Sequence: A sequence is a database object, which can generate unique, sequential integer values.

Synonym: A synonym is a database object, which is used as an alias (alternative name) for a table view or sequence.

5.10 QUESTIONS FOR DISCUSSION

1. How the Schema objects can be created and manipulated? With SQL what type of objects are included in it?
2. Define data integrity and explain the different types of data integrity.
3. What is the processor to store the table data?
4. What are sequences? Explain the different types of indexes.
5. Discuss the various types of privileges given to user in SQL.
6. Explain Synonyms.

Check Your Progress: Model Answers

1. Data
2. data segment
3. smaller
4. access

5.11 SUGGESTED READINGS

Peter Rob, Carlos Coronel, *Database Systems: Design, Implementation and Management*, Seventh edition, Thomson Learning, 2007

Silberschatz, Korth, Sudarshan, *Database System Concepts*, Fifth edition, McGraw-Hill, 2005

Elmasari Navathe, *Fundamentals of Database Systems*, Third edition, Pearson Education Asia, 2001

E. J. Yannakoudakis, *The Architectural Logic of Database Systems*, Springer-Verlag, Digitized 2007

Fred R. McFadden, Jeffrey A. Hoffer, *Database Management*, Benjamin/Cummings, Digitized 2007

Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems*, Third edition, McGraw-Hill Higher Education, 2003

UNIT IV

LESSON

6

PL/SQL

CONTENTS

6.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of PL/SQL
- Discuss how to identify the variables and constants
- Describe the significance of cursors
- Identify and explain the control structures
- Discuss the concept of modularity
- Explain the concept of information hiding
- Discuss the error handling
- Explain the architecture of PL/SQL
- Describe the advantages of PL/SQL
- Explain what's new in PL/SQL
- Explain transactions

6.1 INTRODUCTION

This lesson surveys the main features of PL/SQL and points out the advantages they offer. It also acquaints you with the basic concepts behind PL/SQL and the general appearance of PL/SQL programs. You see how PL/SQL bridges the gap between database technology and procedural programming languages.

6.2 UNDERSTANDING THE MAIN FEATURES OF PL/SQL

A good way to get acquainted with PL/SQL is to look at a sample program. The program below processes an order for a tennis racket. First, it declares a variable of type NUMBER to store the quantity of tennis rackets on hand. Then, it retrieves the quantity on hand from a database table named inventory. If the quantity is greater than zero, the program updates the table and inserts a purchase record into another table named purchase_record. Otherwise, the program inserts an out-of-stock record into the purchase_record table.

```
-- available online in file 'examp1'
DECLARE
  qty_on_hand  NUMBER(5);
BEGIN
  SELECT quantity INTO qty_on_hand FROM inventory
    WHERE product = 'TENNIS RACKET'
    FOR UPDATE OF quantity;
  IF qty_on_hand > 0 THEN  - check quantity
```

```
    UPDATE inventory SET quantity = quantity - 1
      WHERE product = 'TENNIS RACKET';
    INSERT INTO purchase_record
      VALUES ('Tennis racket purchased', SYSDATE);
  ELSE
    INSERT INTO purchase_record
      VALUES ('Out of tennis rackets', SYSDATE);
  END IF;
  COMMIT;
END;
```

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow-of-control statements to process the data. You can also declare constants and variables, define procedures and functions, and trap runtime errors. Thus, PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

Block Structure

PL/SQL is a block-structured language. That is, the basic units (procedures, functions, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. Typically, each logical block corresponds to a problem or subproblem to be solved. Thus, PL/SQL supports the divide-and-conquer approach to problem solving called stepwise refinement.

That way, you can place declarations close to where they are used. The declarations are local to the block and cease to exist when the block completes.

A PL/SQL block has three parts: a declarative part, an executable part, and an exception-handling part. (In PL/SQL, a warning or error condition is called an exception.) Only the executable part is required.

The order of the parts is logical. First comes the declarative part, in which items can be declared. Once declared, items can be manipulated in the executable part. Exceptions raised during execution can be dealt with in the exception-handling part.

You can nest sub-blocks in the executable and exception-handling parts of a PL/SQL block or subprogram but not in the declarative part. Also, you can define local subprograms in the declarative part of any block. However, you can call local subprograms only from the block in which they are defined.

6.3 VARIABLES AND CONSTANTS

PL/SQL lets you declare constants and variables, then use them in SQL and procedural statements anywhere an expression can be used. However, forward references are not allowed. So, you must declare a constant or variable before referencing it in other statements, including other declarative statements.

6.3.1 Declaring Variables

Variables can have any SQL datatype, such as CHAR, DATE, or NUMBER, or any PL/SQL datatype, such as BOOLEAN or BINARY_INTEGER. For example, assume that you want to declare

a variable named part_no to hold 4-digit numbers and a variable named in_stock to hold the Boolean value TRUE or FALSE. You declare these variables as follows:

```
part_no  NUMBER(4);

in_stock BOOLEAN;
```

You can also declare nested tables, variable-size arrays (varrays for short), and records using the TABLE, VARRAY, and RECORD composite datatypes.

Assigning Values to a Variable

You can assign values to a variable in three ways. The first way uses the assignment operator (:=), a colon followed by an equal sign. You place the variable to the left of the operator and an expression (which can include function calls) to the right. A few examples follow:

```
tax := price * tax_rate;

valid_id := FALSE;

bonus := current_salary * 0.10;

wages := gross_pay(emp_id, st_hrs, ot_hrs) - deductions;
```

The second way to assign values to a variable is by selecting (or fetching) database values into it. In the example below, you have Oracle compute a 10% bonus when you select the salary of an employee. Now, you can use the variable bonus in another computation or insert its value into a database table.

```
SELECT sal * 0.10 INTO bonus FROM emp WHERE empno = emp_id;
```

The third way to assign values to a variable is by passing it as an OUT or IN OUT parameter to a subprogram. As the following example shows, an IN OUT parameter lets you pass initial values to the subprogram being called and return updated values to the caller:

```
DECLARE
  my_sal REAL(7,2);
  PROCEDURE adjust_salary (emp_id INT, salary IN OUT REAL) IS ...
BEGIN
  SELECT AVG(sal) INTO my_sal FROM emp;
  adjust_salary(7788, my_sal); -- assigns a new value to my_sal
```

6.3.2 Declaring Constants

Declaring a constant is like declaring a variable except that you must add the keyword CONSTANT and immediately assign a value to the constant. Thereafter, no more assignments to the constant are allowed. In the following example, you declare a constant named credit_limit:

```
credit_limit CONSTANT REAL := 5000.00;
```

6.4 CURSORS

Oracle uses work areas to execute SQL statements and store processing information. A PL/SQL construct called a cursor lets you name a work area and access its stored information. There are two kinds of cursors: implicit and explicit. PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row. For queries that return more than one row, you can explicitly declare a cursor to process the rows individually. An example follows:

```
DECLARE
  CURSOR c1 IS
    SELECT empno, ename, job FROM emp WHERE deptno = 20;
```

The set of rows returned by a multi-row query is called the result set. Its size is the number of rows that meet your search criteria. An explicit cursor "points" to the current row in the result set. This allows your program to process the rows one at a time.

Multi-row query processing is somewhat like file processing. For example, a COBOL program opens a file, processes records, then closes the file. Likewise, a PL/SQL program opens a cursor, processes rows returned by a query, then closes the cursor. Just as a file pointer marks the current position in an open file, a cursor marks the current position in a result set.

You use the OPEN, FETCH, and CLOSE statements to control a cursor. The OPEN statement executes the query associated with the cursor, identifies the result set, and positions the cursor before the first row. The FETCH statement retrieves the current row and advances the cursor to the next row. When the last row has been processed, the CLOSE statement disables the cursor.

6.4.1 Cursor FOR Loops

In most situations that require an explicit cursor, you can simplify coding by using a cursor FOR loop instead of the OPEN, FETCH, and CLOSE statements. A cursor FOR loop implicitly declares its loop index as a record that represents a row fetched from the database. Next, it opens a cursor, repeatedly fetches rows of values from the result set into fields in the record, then closes the cursor when all rows have been processed. In the following example, the cursor FOR loop implicitly declares emp_rec as a record:

```
DECLARE
  CURSOR c1 IS
    SELECT ename, sal, hiredate, deptno FROM emp;
  ...
BEGIN
  FOR emp_rec IN c1 LOOP
    ...
    salary_total := salary_total + emp_rec.sal;
  END LOOP;
```

To reference individual fields in the record, you use dot notation, in which a dot (.) serves as the component selector.

6.4.2 Cursor Variables

Like a cursor, a cursor variable points to the current row in the result set of a multi-row query. But, unlike a cursor, a cursor variable can be opened for any type-compatible query. It is not tied to a specific query. Cursor variables are true PL/SQL variables, to which you can assign new values and which you can pass to subprograms stored in an Oracle database. This gives you more flexibility and a convenient way to centralize data retrieval.

Typically, you open a cursor variable by passing it to a stored procedure that declares a cursor variable as one of its formal parameters. The following procedure opens the cursor variable generic_cv for the chosen query:

```
PROCEDURE open_cv (generic_cv IN OUT GenericCurTyp,choice NUMBER) IS
BEGIN
  IF choice = 1 THEN
    OPEN generic_cv FOR SELECT * FROM emp;
  ELSIF choice = 2 THEN
    OPEN generic_cv FOR SELECT * FROM dept;
  ELSIF choice = 3 THEN
    OPEN generic_cv FOR SELECT * FROM salgrade;
END IF;
  ...
END;
```

6.4.3 Attributes

PL/SQL variables and cursors have attributes, which are properties that let you reference the datatype and structure of an item without repeating its definition. Database columns and tables have similar attributes, which you can use to ease maintenance. A percent sign (%) serves as the attribute indicator.

%TYPE

The %TYPE attribute provides the datatype of a variable or database column. This is particularly useful when declaring variables that will hold database values. For example, assume there is a column named title in a table named books. To declare a variable named my_title that has the same datatype as column title, use dot notation and the %TYPE attribute, as follows:

```
my_title books.title%TYPE;
```

Declaring my_title with %TYPE has two advantages. First, you need not know the exact datatype of title. Second, if you change the database definition of title (make it a longer character string for example), the datatype of my_title changes accordingly at run time.

%ROWTYPE

In PL/SQL, records are used to group data. A record consists of a number of related fields in which data values can be stored. The %ROWTYPE attribute provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable.

Columns in a row and corresponding fields in a record have the same names and datatypes. In the example below, you declare a record named dept_rec. Its fields have the same names and datatypes as the columns in the dept table.

6.4.4 Declare

```
dept_rec dept%ROWTYPE;  -- declare record variable
```

You use dot notation to reference fields, as the following example shows:

```
my_deptno := dept_rec.deptno;
```

If you declare a cursor that retrieves the last name, salary, hire date, and job title of an employee, you can use %ROWTYPE to declare a record that stores the same information, as follows:

```
DECLARE
  CURSOR c1 IS
    SELECT ename, sal, hiredate, job FROM emp;
  emp_rec c1%ROWTYPE;  -- declare record variable that represents
                       -- a row fetched from the emp table
```

When you execute the statement

```
FETCH c1 INTO emp_rec;
```

the value in the ename column of the emp table is assigned to the ename field of emp_rec, the value in the sal column is assigned to the sal field, and so on.

6.5 CONTROL STRUCTURES

Control structures are the most important PL/SQL extension to SQL. Not only does PL/SQL let you manipulate Oracle data, it lets you process the data using conditional, iterative, and sequential flow-of-control statements such as IF-THEN-ELSE, CASE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN, and GOTO. Collectively, these statements can handle any situation.

6.5.1 Conditional Control

Often, it is necessary to take alternative actions depending on circumstances. The IF-THEN-ELSE statement lets you execute a sequence of statements conditionally. The IF clause checks a condition; the THEN clause defines what to do if the condition is true; the ELSE clause defines what to do if the condition is false or null.

Consider the program below, which processes a bank transaction. Before allowing you to withdraw $500 from account 3, it makes sure the account has sufficient funds to cover the withdrawal. If the

funds are available, the program debits the account. Otherwise, the program inserts a record into an audit table.

```
-- available online in file 'examp2'
DECLARE
  acct_balance NUMBER(11,2);
  acct        CONSTANT NUMBER(4) := 3;
  debit_amt   CONSTANT NUMBER(5,2) := 500.00;
BEGIN

  SELECT bal INTO acct_balance FROM accounts
    WHERE account_id = acct
    FOR UPDATE OF bal;
  IF acct_balance >= debit_amt THEN
    UPDATE accounts SET bal = bal - debit_amt
      WHERE account_id = acct;
  ELSE
    INSERT INTO temp VALUES
      (acct, acct_balance, 'Insufficient funds');
        -- insert account, current balance, and message
  END IF;
  COMMIT;
END;
```

To choose among several values or courses of action, you can use CASE constructs. The CASE expression evaluates a condition and returns a value for each case. The case statement evaluates a condition and performs an action (which might be an entire PL/SQL block) for each case.

```
-- This CASE statement performs different actions based
-- on a set of conditional tests.
CASE
 WHEN shape = 'square' THEN area := side * side;
 WHEN shape = 'circle' THEN
  BEGIN
    area := pi * (radius * radius);
    DBMS_OUTPUT.PUT_LINE('Value is not exact because pi is irrational.');
```

```
  END;
 WHEN shape = 'rectangle' THEN area := length * width;
 ELSE
  BEGIN
   DBMS_OUTPUT.PUT_LINE('No formula to calculate area of a' || shape);
   RAISE PROGRAM_ERROR;
  END;
END CASE;
```

A sequence of statements that uses query results to select alternative actions is common in database applications. Another common sequence inserts or deletes a row only if an associated entry is found in another table. You can bundle these common sequences into a PL/SQL block using conditional logic.

6.5.2 Iterative Control

LOOP statements let you execute a sequence of statements multiple times. You place the keyword LOOP before the first statement in the sequence and the keywords END LOOP

after the last statement in the sequence. The following example shows the simplest kind of loop, which repeats a sequence of statements continually:

```
LOOP
  -- sequence of statements
END LOOP;
```

The FOR-LOOP statement lets you specify a range of integers, then execute a sequence of statements once for each integer in the range. For example, the following loop inserts 500 numbers and their square roots into a database table:

```
FOR num IN 1..500 LOOP
  INSERT INTO roots VALUES (num, SQRT(num));
END LOOP;
```

The WHILE-LOOP statement associates a condition with a sequence of statements. Before each iteration of the loop, the condition is evaluated. If the condition is true, the sequence of statements is executed, then control resumes at the top of the loop. If the condition is false or null, the loop is bypassed and control passes to the next statement.

In the following example, you find the first employee who has a salary over $2500 and is higher in the chain of command than employee 7499:

```
-- available online in file 'examp3'
DECLARE
  salary      emp.sal%TYPE := 0;
  mgr_num     emp.mgr%TYPE;
```

```
  last_name     emp.ename%TYPE;
  starting_empno emp.empno%TYPE := 7499;
BEGIN
  SELECT mgr INTO mgr_num FROM emp
    WHERE empno = starting_empno;
  WHILE salary <= 2500 LOOP
    SELECT sal, mgr, ename INTO salary, mgr_num, last_name
      FROM emp WHERE empno = mgr_num;
  END LOOP;
  INSERT INTO temp VALUES (NULL, salary, last_name);
  COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    INSERT INTO temp VALUES (NULL, NULL, 'Not found');
    COMMIT;
END;
```

The EXIT-WHEN statement lets you complete a loop if further processing is impossible or undesirable. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement. In the following example, the loop completes when the value of total exceeds 25,000.

```
LOOP
  ...
  total := total + salary;
  EXIT WHEN total > 25000;  -- exit loop if condition is true
END LOOP;
-- control resumes here
```

6.5.3 Sequential Control

The GOTO statement lets you branch to a label unconditionally. The label, an undeclared identifier enclosed by double angle brackets, must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block, as the following example shows:

```
IF rating > 90 THEN
  GOTO calc_raise;  -- branch to label
```

```
END IF;
...
<<calc_raise>>
IF job_title = 'SALESMAN' THEN -- control resumes here
  amount := commission * 0.25;
ELSE
  amount := salary * 0.10;
END IF;
```

6.6 MODULARITY

Modularity lets you break an application down into manageable, well-defined modules. Through successive refinement, you can reduce a complex problem to a set of simple problems that have easy-to-implement solutions. PL/SQL meets this need with program units, which include blocks, subprograms, and packages.

6.6.1 Subprograms

PL/SQL has two types of subprograms called procedures and functions, which can take parameters and be invoked (called). As the following example shows, a subprogram is like a miniature program, beginning with a header followed by an optional declarative part, an executable part, and an optional exception-handling part:

```
PROCEDURE award_bonus (emp_id NUMBER) IS
  bonus      REAL;
  comm_missing EXCEPTION;
BEGIN -- executable part starts here
  SELECT comm * 0.15 INTO bonus FROM emp WHERE empno = emp_id;
  IF bonus IS NULL THEN
    RAISE comm_missing;
  ELSE
    UPDATE payroll SET pay = pay + bonus WHERE empno = emp_id;
  END IF;
EXCEPTION -- exception-handling part starts here
  WHEN comm_missing THEN
    ...
END award_bonus;
```

When called, this procedure accepts an employee number. It uses the number to select the employee's commission from a database table and, at the same time, compute a 15% bonus. Then, it checks the bonus amount. If the bonus is null, an exception is raised; otherwise, the employee's payroll record is updated.

6.6.2 Packages

PL/SQL lets you bundle logically related types, variables, cursors, and subprograms into a package. Each package is easy to understand and the interfaces between packages are simple, clear, and well defined. This aids application development.

Packages usually have two parts: a specification and a body. The specification is the interface to your applications; it declares the types, constants, variables, exceptions, cursors, and subprograms available for use. The body defines cursors and subprograms and so implements the specification.

In the following example, you package two employment procedures:

```
CREATE PACKAGE emp_actions AS  -- package specification
  PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...);
  PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;
CREATE PACKAGE BODY emp_actions AS  -- package body
  PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...) IS
  BEGIN
    INSERT INTO emp VALUES (empno, ename, ...);
  END hire_employee;
  PROCEDURE fire_employee (emp_id NUMBER) IS
  BEGIN
    DELETE FROM emp WHERE empno = emp_id;
  END fire_employee;
END emp_actions;
```

Only the declarations in the package specification are visible and accessible to applications. Implementation details in the package body are hidden and inaccessible.

Packages can be compiled and stored in an Oracle database, where their contents can be shared by many applications. When you call a packaged subprogram for the first time, the whole package is loaded into memory. So, subsequent calls to related subprograms in the package require no disk I/O. Thus, packages can enhance productivity and improve performance.

6.7 INFORMATION HIDING

With information hiding, you see only the details that are relevant at a given level of algorithm and data structure design. Information hiding keeps high-level design decisions separate from low-level design details, which are more likely to change.

Algorithms

You implement information hiding for algorithms through top-down design. Once you define the purpose and interface specifications of a low-level procedure, you can ignore the implementation details. They are hidden at higher levels. For example, the implementation of a procedure named raise_salary is hidden. All you need to know is that the procedure will increase a specific employee's salary by a given amount. Any changes to the definition of raise_salary are transparent to calling applications.

Data Structures

You implement information hiding for data structures though data encapsulation. By developing a set of utility subprograms for a data structure, you insulate it from users and other developers. That way, other developers know how to use the subprograms that operate on the data structure but not how the structure is represented.

With PL/SQL packages, you can specify whether subprograms are public or private. Thus, packages enforce data encapsulation by letting you put subprogram definitions in a black box. A private definition is hidden and inaccessible. Only the package, not your application, is affected if the definition changes. This simplifies maintenance and enhancement.

6.8 ERROR HANDLING

PL/SQL makes it easy to detect and process predefined and user-defined error conditions called exceptions. When an error occurs, an exception is raised. That is, normal execution stops and control transfers to the exception-handling part of your PL/SQL block or subprogram. To handle raised exceptions, you write separate routines called exception handlers.

Predefined exceptions are raised implicitly by the runtime system. For example, if you try to divide a number by zero, PL/SQL raises the predefined exception ZERO_DIVIDE automatically. You must raise user-defined exceptions explicitly with the RAISE statement.

You can define exceptions of your own in the declarative part of any PL/SQL block or subprogram. In the executable part, you check for the condition that needs special attention. If you find that the condition exists, you execute a RAISE statement. In the example below, you compute the bonus earned by a salesperson. The bonus is based on salary and commission. So, if the commission is null, you raise the exception comm_missing.

```
DECLARE
  ...
  comm_missing EXCEPTION;  -- declare exception
BEGIN
  ...
  IF commission IS NULL THEN
    RAISE comm_missing;  -- raise exception
  END IF;
  bonus := (salary * 0.10) + (commission * 0.15);
```

```
EXCEPTION
  WHEN comm_missing THEN ... -- process the exception
```

6.9 PL/SQL ARCHITECTURE

The PL/SQL compilation and run-time system is a technology, not an independent product. Think of this technology as an engine that compiles and executes PL/SQL blocks and subprograms. The engine can be installed in an Oracle server or in an application development tool such as Oracle Forms or Oracle Reports. So, PL/SQL can reside in two environments:

6.9.1 The Oracle Database Server

Oracle Tools

These two environments are independent. PL/SQL is bundled with the Oracle server but might be unavailable in some tools. In either environment, the PL/SQL engine accepts as input any valid PL/SQL block or subprogram. The engine executes procedural statements but sends SQL statements to the SQL Statement Executor in the Oracle server.

In the Oracle Database Server

Application development tools that lack a local PL/SQL engine must rely on Oracle to process PL/SQL blocks and subprograms. When it contains the PL/SQL engine, an Oracle server can process PL/SQL blocks and subprograms as well as single SQL statements. The Oracle server passes the blocks and subprograms to its local PL/SQL engine.

Anonymous Blocks

Anonymous PL/SQL blocks can be embedded in an Oracle Precompiler or OCI program. At run time, the program, lacking a local PL/SQL engine, sends these blocks to the Oracle server, where they are compiled and executed. Likewise, interactive tools such as SQL*Plus and Enterprise Manager, lacking a local PL/SQL engine, must send anonymous blocks to Oracle.

6.9.2 Stored Subprograms

Subprograms can be compiled separately and stored permanently in an Oracle database, ready to be executed. A subprogram explicitly CREATEd using an Oracle tool is called a stored subprogram. Once compiled and stored in the data dictionary, it is a schema object, which can be referenced by any number of applications connected to that database.

Stored subprograms defined within a package are called packaged subprograms. Those defined independently are called standalone subprograms. Those defined within another subprogram or within a PL/SQL block are called local subprograms, which cannot be referenced by other applications and exist only for the convenience of the enclosing block.

Stored subprograms offer higher productivity, better performance, memory savings, application integrity, and tighter security. For example, by designing applications around a library of stored procedures and functions, you can avoid redundant coding and increase your productivity.

You can call stored subprograms from a database trigger, another stored subprogram, an Oracle Precompiler application, an OCI application, or interactively from SQL*Plus or Enterprise Manager. For example, you might call the standalone procedure create_dept from SQL*Plus as follows:

```
SQL> CALL create_dept('FINANCE', 'NEW YORK');
```

Subprograms are stored in parsed, compiled form. So, when called, they are loaded and passed to the PL/SQL engine immediately. Also, they take advantage of shared memory. So, only one copy of a subprogram need be loaded into memory for execution by multiple users.

Database Triggers

A database trigger is a stored subprogram associated with a database table, view, or event. For instance, you can have Oracle fire a trigger automatically before or after an INSERT, UPDATE, or DELETE statement affects a table. One of the many uses for database triggers is to audit data modifications. For example, the following table-level trigger fires whenever salaries in the emp table are updated:

```
CREATE TRIGGER audit_sal
  AFTER UPDATE OF sal ON emp
  FOR EACH ROW
BEGIN
  INSERT INTO emp_audit VALUES ...
END;
```

The executable part of a trigger can contain procedural statements as well as SQL data manipulation statements. Besides table-level triggers, there are instead-of triggers for views and system-event triggers for schemas.

In Oracle Tools

When it contains the PL/SQL engine, an application development tool can process PL/SQL blocks and subprograms. The tool passes the blocks to its local PL/SQL engine. The engine executes all procedural statements at the application site and sends only SQL statements to Oracle. Thus, most of the work is done at the application site, not at the server site.

Furthermore, if the block contains no SQL statements, the engine executes the entire block at the application site. This is useful if your application can benefit from conditional and iterative control.

Frequently, Oracle Forms applications use SQL statements merely to test the value of field entries or to do simple computations. By using PL/SQL instead, you can avoid calls to the Oracle server. Moreover, you can use PL/SQL functions to manipulate field entries.

6.10 ADVANTAGES OF PL/SQL

PL/SQL is a completely portable, high-performance transaction processing language that offers the following advantages:

- Support for SQL
- Support for object-oriented programming

- Better performance
- Higher productivity
- Full portability
- Tight integration with Oracle
- Tight security

Support for SQL

SQL has become the standard database language because it is flexible, powerful, and easy to learn. A few English-like commands such as SELECT, INSERT, UPDATE, and DELETE make it easy to manipulate the data stored in a relational database.

SQL is non-procedural, meaning that you can state what you want done without stating how to do it. Oracle determines the best way to carry out your request. There is no necessary connection between consecutive statements because Oracle executes SQL statements one at a time.

PL/SQL lets you use all the SQL data manipulation, cursor control, and transaction control commands, as well as all the SQL functions, operators, and pseudocolumns. So, you can manipulate Oracle data flexibly and safely. Also, PL/SQL fully supports SQL datatypes. That reduces the need to convert data passed between your applications and the database.

PL/SQL also supports dynamic SQL, an advanced programming technique that makes your applications more flexible and versatile. Your programs can build and process SQL data definition, data control, and session control statements "on the fly" at run time.

Support for Object-Oriented Programming

Object types are an ideal object-oriented modeling tool, which you can use to reduce the cost and time required to build complex applications. Besides allowing you to create software components that are modular, maintainable, and reusable, object types allow different teams of programmers to develop software components concurrently.

By encapsulating operations with data, object types let you move data-maintenance code out of SQL scripts and PL/SQL blocks into methods. Also, object types hide implementation details, so that you can change the details without affecting client programs.

In addition, object types allow for realistic data modeling. Complex real-world entities and relationships map directly into object types. That helps your programs better reflect the world they are trying to simulate.

Better Performance

Without PL/SQL, Oracle must process SQL statements one at a time. Each SQL statement results in another call to Oracle and higher performance overhead. In a networked environment, the overhead can become significant. Every time a SQL statement is issued, it must be sent over the network, creating more traffic.

However, with PL/SQL, an entire block of statements can be sent to Oracle at one time. This can drastically reduce communication between your application and Oracle. If your application is database intensive, you can use PL/SQL blocks and subprograms to group SQL statements before sending them to Oracle for execution.

PL/SQL stored procedures are compiled once and stored in executable form, so procedure calls are quick and efficient. Also, stored procedures, which execute in the server, can be invoked over slow network connections with a single call. That reduces network traffic and improves round-trip response times. Executable code is automatically cached and shared among users. That lowers memory requirements and invocation overhead.

PL/SQL also improves performance by adding procedural processing power to Oracle tools. Using PL/SQL, a tool can do any computation quickly and efficiently without calling on the Oracle server. This saves time and reduces network traffic.

Higher Productivity

PL/SQL adds functionality to non-procedural tools such as Oracle Forms and Oracle Reports. With PL/SQL in these tools, you can use familiar procedural constructs to build applications. For example, you can use an entire PL/SQL block in an Oracle Forms trigger. You need not use multiple trigger steps, macros, or user exits. Thus, PL/SQL increases productivity by putting better tools in your hands.

Also, PL/SQL is the same in all environments. As soon as you master PL/SQL with one Oracle tool, you can transfer your knowledge to other tools, and so multiply the productivity gains. For example, scripts written with one tool can be used by other tools.

Full Portability

Applications written in PL/SQL are portable to any operating system and platform on which Oracle runs. In other words, PL/SQL programs can run anywhere Oracle can run; you need not tailor them to each new environment. That means you can write portable program libraries, which can be reused in different environments.

Tight Integration with SQL

The PL/SQL and SQL languages are tightly integrated. PL/SQL supports all the SQL datatypes and the non-value NULL. That allows you manipulate Oracle data easily and efficiently. It also helps you to write high-performance code.

The %TYPE and %ROWTYPE attributes further integrate PL/SQL with SQL. For example, you can use the %TYPE attribute to declare variables, basing the declarations on the definitions of database columns. If a definition changes, the variable declaration changes accordingly the next time you compile or run your program. The new definition takes effect without any effort on your part. This provides data independence, reduces maintenance costs, and allows programs to adapt as the database changes to meet new business needs.

Tight Security

PL/SQL stored procedures enable you to partition application logic between the client and server. That way, you can prevent client applications from manipulating sensitive Oracle data. Database triggers written in PL/SQL can disable application updates selectively and do content-based auditing of user inserts.

Furthermore, you can restrict access to Oracle data by allowing users to manipulate it only through stored procedures that execute with their definer's privileges. For example, you can grant users access to a procedure that updates a table, but not grant them access to the table itself.

6.11 WHAT'S NEW IN PL/SQL?

This section describes new features of PL/SQL release 9.0.1 and provides pointers to additional information.

You can now insert into or update a SQL table by specifying a PL/SQL record variable, rather than specifying each record attribute separately. You can also select entire rows into a PL/SQL table of records, rather than using a separate PL/SQL table for each SQL column.

Associative Arrays

You can create collections that are indexed by VARCHAR2 values, providing features similar to hash tables in Perl and other languages.

User-defined constructors

You can now override the system default constructor for an object type with your own function.

Enhancements to UTL_FILE package

UTL_FILE contains several new functions that let you perform general file-management operations from PL/SQL.

TREAT Function for Object Types

You can dynamically choose the level of type inheritance to use when calling object methods. That is, you can reference an object type that inherits from several levels of parent types, and call a method from a specific parent type. This function is similar to the SQL function of the same name.

Integration of SQL and PL/SQL Parsers

PL/SQL now supports the complete range of syntax for SQL statements, such as INSERT, UPDATE, DELETE, and so on. If you received errors for valid SQL syntax in PL/SQL programs before, those statements should now work.

Because of more consistent error-checking, you might find that some invalid code is now found at compile time instead of producing an error at runtime, or vice versa. You might need to change the source code as part of the migration procedure.

CASE Statements and Expressions

CASE statements and expressions are a shorthand way of representing IF/THEN choices with multiple alternatives.

Inheritance and Dynamic Method Dispatch

Types can be declared in a supertype/subtype hierarchy, with subtypes inheriting attributes and methods from their supertypes. The subtypes can also add new attributes and methods, and override existing methods. A call to an object method executes the appropriate version of the method, based on the type of the object.

Type Evolution

Attributes and methods can be added to and dropped from object types, without the need to re-create the types and corresponding data. This feature lets the type hierarchy adapt to changes in the application, rather than being planned out entirely in advance.

New Date/Time Types

The new datatype TIMESTAMP records time values including fractional seconds. New datatypes TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE allow you to adjust date and time values to account for time zone differences. You can specify whether the time zone observes daylight savings time, to account for anomalies when clocks shift forward or backward. New datatypes INTERVAL DAY TO SECOND and INTERVAL YEAR TO MONTH represent differences between two date and time values, simplifying date arithmetic.

Native Compilation of PL/SQL Code

Improve performance by compiling Oracle-supplied and user-written stored procedures into native executables, using typical C development tools. This setting is saved so that the procedure is compiled the same way if it is later invalidated.

Improved Globalization and National Language Support

Data can be stored in Unicode format using fixed-width or variable-width character sets. String handling and storage declarations can be specified using byte lengths, or character lengths where the number of bytes is computed for you. You can set up the entire database to use the same length semantics for strings, or specify the settings for individual procedures; this setting is remembered if a procedure is invalidated.

Table Functions and Cursor Expressions

You can query a set of returned rows like a table. Result sets can be passed from one function to another, letting you set up a sequence of transformations with no table to hold intermediate results. Rows of the result set can be returned a few at a time, reducing the memory overhead for producing large result sets within a function.

Multilevel Collections

You can nest the collection types, for example to create a VARRAY of PL/SQL tables, a VARRAY of VARRAYs, or a PL/SQL table of PL/SQL tables. You can model complex data structures such as multidimensional arrays in a natural way.

Better Integration for LOB Datatypes

You can operate on LOB types much like other similar types. You can use character functions on CLOB and NCLOB types. You can treat BLOB types as RAWs. Conversions between LOBs and other types are much simpler, particularly when converting from LONG to LOB types.

Enhancements to Bulk Operations

You can now perform bulk SQL operations, such as bulk fetches, using native dynamic SQL (the EXECUTE IMMEDIATE statement). You can perform bulk insert or update operations that continue despite errors on some rows, then examine the problems after the operation is complete.

MERGE Statement

This specialized statement combines insert and update into a single operation. It is intended for data warehousing applications that perform particular patterns of inserts and updates.

6.12 TRANSACTION

Transaction is a logical unit of data manipulation related tasks wherein either the all the component tasks must be completed or none of them is executed in order to keep the database consistent. When many transactions proceed in the database environment it is imperative that a strict control is applied on them failing which the consistency of the database cannot be ensured.

Transactions can very easily cause undesirable inconsistencies in a database especially when many of them are executing concurrently. The problem has been studied in detail by designers and experts and it has been concluded that if a database enforces certain conditions on transaction management the problem of inconsistency can be avoided.

The term ACID is abbreviation for those properties that must be associated with transactions so that the integrity of the database is ensured. The term when extended reads:

A: Atomicity

C: Consistency

I: Isolation

D: Durability

Atomicity

A transaction typically contains a number of database operations. The Atomicity property of a transaction ensures that either all the operations are carried out successfully or none of the operations is carried out at all. In the former case the transaction is said to have completed successfully and in the later it is said to have failed. A failed transaction does not have any effect on the state of the database. To monitor and control atomicity property of transactions database systems provide a components called transaction management system.

Consistency

This property of a transaction requires that the integrity rules of a database must not be violated. For example, if an amount is being transferred from one relation to another relation then the consistency rule must ensure that the transaction does not create or destroy an additional amount during its operation. That is the total amount in both the relations must remain the same. This property ensures that the database remains consistent before and after the execution of a transaction. The onus to enforce consistency property lies with the application programmers.

Isolation

In a multi-transaction environment many transactions may be executing concurrently on a single database. This property makes sure that every transaction executes independent of each other. To monitor and control atomicity property of transactions database systems provide a component called concurrency control system.

Durability

This property ensures that the changes made to the database are recorded in the physical database on successful completion of a transaction. The recovery management component of the database management system takes care of durability of the transactions.

To elucidate ACID properties consider a transaction (T1) in a banking database system with following characteristics:

- The database is maintained on a secondary storage device like a hard disk.
- The system has a number of accounts.
- Transactions can read a data from the database to a variable in the memory by a READ operation - READ(A,B) meaning the value of A (a database field) is copied into the variable B in the memory.
- Transactions can write a data stored in a variable A into the database by a WRITE operation - WRITE(A,B) meaning the current value stored in variable A is copied into B in the database immediately.
- The transaction T1 transfers Rs. 200 from an account named FirstAc to another account named SecondAc, i.e.,

```
T1:     READ(FirstAc.Balance,X)
        X ← X-200
        WRITE (X,FirstAc.Balance
        X ← X+200
        READ(SecondAc.Balance,X)
        WRITE (X,SecondAc.Balance)
```

In execution of this transaction the system must ensure that the sum FirstAc.Balance + SecondAc.Balance must not change failing which would mean that some amount of money has either created or destroyed spuriously. This is the consistency property of this transaction. The application programmer writing this transaction must enforce this consistency condition.

Now, consider that the transaction fails in between due to power failure or any other reason after the FirstAc.Balance was decreased by 200 but before SecondAc.Balance was updated. The consistency rule is violated. This leaves that database in consistent state. The transaction management component of the database management system applies mechanism to ensure that the transaction follows atomicity property so that either both the values are updated or neither.

We were assuming here that this is the only transaction being executed at this time. However, in a real life situation many transactions execute simultaneously. Now, suppose after the FirstAc.Balance has been updated and before SecondAc.Balance is updated another transaction T2 accesses SecondAc.Balance and adds 400 into it; then again the SecondAc.Balance is updated. At this point the database again will enter into an inconsistent state. To avoid this concurrency control component of the database management system employs mechanisms to ensure isolation of the transaction through various concurrency control techniques.

Another assumption made here is that the WRITE operation updates the database immediately. However, in practice due to many reasons updation is not done right away. The operations are performed on a copy of the database and the actual recording in the database is done much later. The recovery management component of the database management system applies the property of durability on the transaction to make sure that the changes created by transactions are eventually recorded in the database.

Check Your Progress

Fill in the blanks:

1. A block (or sub-block) lets you group logically related declarations and ..... .
2. A PL/SQL construct called a cursor lets you name a work area and ..... its stored information.
3. PL/SQL now supports the complete range of syntax for ..... statements
4. A database trigger is a stored subprogram associated with a ....., view, or event.
5. Predefined exceptions are raised ..... by the runtime system.

6.13 LET US SUM UP

A good way to get acquainted with PL/SQL is to look at a sample program. The program below processes an order for a tennis racket. First, it declares a variable of type NUMBER to store the quantity of tennis rackets on hand. PL/SQL lets you declare constants and variables, then use them in SQL and procedural statements anywhere an expression can be used. However, forward references are not allowed. Oracle uses work areas to execute SQL statements and store processing information. A PL/SQL construct called a cursor lets you name a work area and access its stored information. Control structures are the most important PL/SQL extension to SQL. Modularity lets you break an application down into manageable, well-defined modules. With information hiding, you see only the details that are relevant at a given level of algorithm and data structure design. PL/SQL makes it easy to detect and process predefined and user-defined error conditions called exceptions. When an error occurs, an exception is raised. The PL/SQL compilation and run-time system is a technology, not an independent product. Think of this technology as an engine that compiles and executes PL/SQL blocks and subprograms. Transaction is a logical unit of data manipulation related tasks wherein either the all the component tasks must be completed or none of them is executed in order to keep the database consistent.

6.14 KEYWORDS

PL/SQL: PL/SQL is a block-structured language.

Cursors: A PL/SQL construct called a cursor lets you name a work area and access its stored information.

Data Triggers: A database trigger is a stored subprogram associated with a database table, view, or event.

Modularity: Modularity lets you break an application down into manageable, well-defined modules.

Transaction: Transactions can very easily cause undesirable inconsistencies in a database especially when many of them are executing concurrently.

6.15 QUESTIONS FOR DISCUSSION

1. Explain the declaration of variables and constants in PL/SQL.
2. What is difference between cursors and cursors variables?

3. Discuss the three types of control structures.
4. What is information hiding in PL/SQL?
5. Explain the PL/SQL architecture. Give its five advantages.
6. Discuss the term transaction? Explain ACID.

Check Your Progress: Model Answers

1. Statements
2. Access
3. SQL
4. database table
5. implicitly

6.16 SUGGESTED READINGS

Peter Rob, Carlos Coronel, *Database Systems: Design, Implementation and Management*, Seventh edition, Thomson Learning, 2007

Silberschatz, Korth, Sudarshan, *Database System Concepts*, Fifth edition, McGraw-Hill, 2005

Elmasari Navathe, *Fundamentals of Database Systems*, Third edition, Pearson Education Asia, 2001

E. J. Yannakoudakis, *The Architectural Logic of Database Systems*, Springer-Verlag, Digitized 2007

Fred R. McFadden, Jeffrey A. Hoffer, *Database Management*, Benjamin/Cummings, Digitized 2007

Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems*, Third edition, McGraw-Hill Higher Education, 2003

LESSON

7

TRIGGERS

CONTENTS

7.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of database triggers
- Discuss different types of triggers
- Describe how to create triggers

7.1 INTRODUCTION

A database trigger is a stored procedure that is fired when an insert, update, or delete statements is issued against the associate table. The name trigger is appropriate, as these are triggered (fired) whenever the above-mentioned commands are executed. A trigger defines an action the database should take when some database-related event occurs. Triggers may used to supplement declarative referential integrity, to enforce complex business rules, or to audit changes to data. The code within a trigger, called the trigger body is made up of PL/SQL blocks. Using triggers is one of the most practical ways to implement routines, thus granting integrity of data or operations.

7.2 DATABASE TRIGGERS

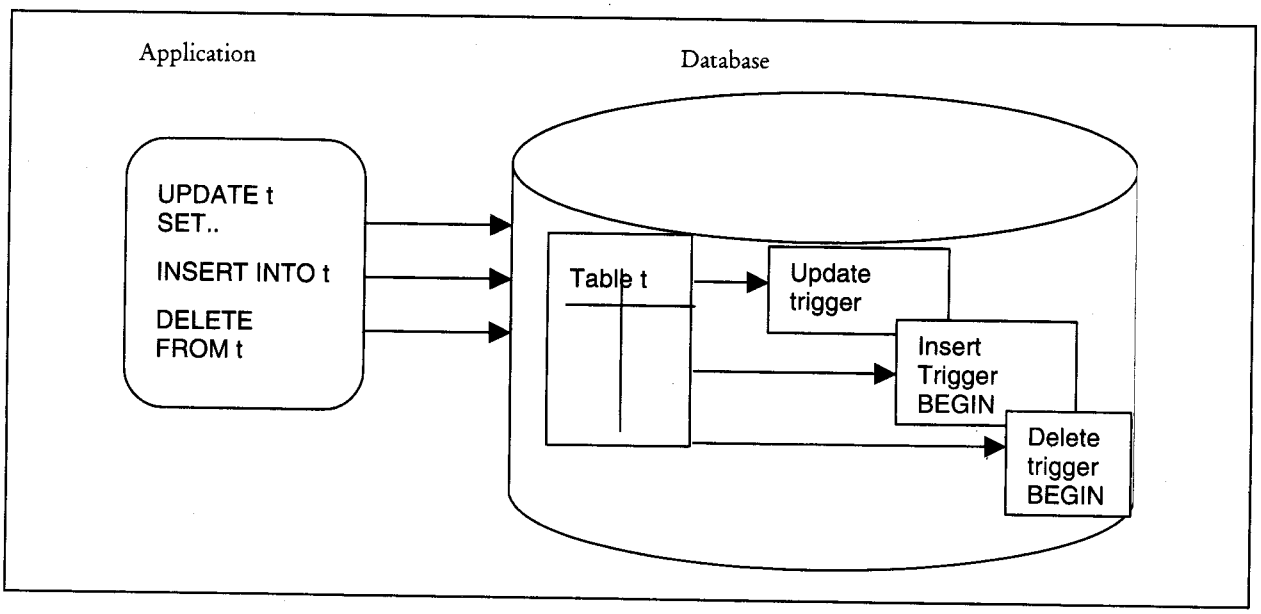
A trigger is automatically executed without any action required by the user. A stored procedure on other hand needs to be explicitly invoked. This is the main difference between a trigger and a stored procedure.

Database triggers can be used for the following purposes :

- To generate data automatically.
- To enforce complex integrity constraints. (e.g. checking with sysdate, checking with data in another table).
- To customize complex security authorizations.
- To maintain replicate tables.
- To audit data modifications.

Required System Privileges

To create a trigger on a table you must be able to alter that table. Therefore, you must either be the owner of that table, have the ALTER TABLE privilege from the owner of the table or you should have the ALTER ANY TABLE system privilege. In addition you must have CREATE TRIGGER system privilege.



Restrictions

A trigger cannot execute the COMMIT, ROLLBACK, or SAVEPOINT commands. It also cannot call procedures or functions that execute those tasks. The SELECT command can be used only with the INTO clause.

A row level (will be dealt in types of trigger) cannot read or change the contents of a table that is being modified. This type of table is one in which the contents are being changed by an INSERT, UPDATE, and DELETE commands, and the command has not been completed.

7.2.1 Components/Parts of a Trigger

A trigger has three parts:

- A trigger statement
- A trigger body
- A trigger restriction

A trigger statement: (SQL command that activates the trigger (triggering event) The trigger can be activated by a SQL command or by a user event. In a table, it can be triggered b the INSERT, UPDATE, or DELETE commands. The INSERT, UPDATE, or DELETE commands enable a trigger. The same trigger can be invoked in more than one situation. The trigger statement fires the trigger body. It also specifies the table to which the trigger is associated.

Trigger Body/ Trigger Action: Trigger body is a PL/SQL block or Java or C routine that is executed when a triggering statement is issued.

Trigger Restriction: Restrictions on a trigger can be achieved using the WHEN clause. They can be included in the definition of a trigger, wherein it specifies what condition must be true for the trigger to be triggered.

Syntax for creating a Trigger: Create or replace Trigger <trigger name>

[before | after] [Insert | Update | Delete] on < table name > [for each statement / for each row] [when < condition >] ;

7.2.2 Types of Triggers

A trigger's type is defined by the type of triggering transaction and by the level at which the trigger is executed. Oracle 9i has the following types of triggers depending on the different applications.

- Before (INSERT, UPDATE, DELETE) Trigger
- After (INSERT, UPDATE, DELETE) Trigger
- Row level Trigger
- Statement level Trigger
- Instead of Trigger
- Schema Trigger
- Database level Trigger

Row Level Trigger

Row level triggers execute once for each row in a transaction. The commands of row level triggers are executed on all rows that are affected by the command that enables the trigger. Row level triggers are the most common type of trigger used, often-used in data auditing applications.

Row level triggers are created using the for each row clause in the create trigger command.

Statement Level Trigger

Statement level triggers are triggered only once for each transaction. For example when an UPDATE command update 15 rows, the commands contained in the trigger are executed only once, and not with every processed row.

Statement level trigger are the default types of trigger created via the create trigger command.

Before and After Trigger

Since triggers are executed by events, they may be set to occur immediately before of after those events. When a trigger is defined, you can specify whether the trigger must occur before or after the triggering event i.e. insert, update, or delete commands.

The BEFORE trigger is used in situations where the trigger action could determine if the trigger itself should be executed, or when you need some preprocessing before executing the command. The AFTER trigger is triggered only after the execution of the associated triggering command.

A table can contain up to 12 triggers associated with the activation commands and triggering event. There are six row-level and six statement-level triggers.

- BEFORE INSERT row / statement
- AFTER INSERT row / statement
- BEFORE UPDATE row / statement
- AFTER UPDATE row / statement
- BEFORE DELETE row / statement
- AFTER DELETE row / statement

Instead of Trigger

Instead of trigger was first featured in Oracle 8. This was something new in the world of triggers. These are triggers that are defined on a view rather than on a table. Such triggers can be used to overcome the restrictions placed by Oracle on any view, which is deemed to be non-updateable. You can use INSTEAD OF trigger to tell Oracle what to do instead of performing the actions that invoked the trigger. For example you can use an INSTAED OF trigger on view to redirect insert into a table or to update multiple tables that re part of a view. You can use INSTEAD OF trigger on either object view or relational view.

In Oracle 8, INSTEAD OF triggers are defined on the same events as their table counterparts: INSERT, UPDATE, or DELETE. Since there is no provision for a trigger, which is run at a lock time, then either locking must be implicit or the application must know what objects to lock. Despite this minor constraint new trigger have removed major constraints on design.

There are a few restrictions on INSTEAD OF trigger. They are available only at the row level and not at the statement level. They can be applied only to views and not to tables.

Database – Level Trigger

You can create triggers to be fired on database events, including errors, logons, logoffs, shutdowns, and startups. You can use this type of trigger to automate database maintenance or auditing actions.

Tom Dick and Harry sales Inc finds that inserting simultaneously into two tables is a very useful job. It achieves what could be done as a series of commands in a single command. They decide to use instead of triggers and create a view on the tables order_master and order_detail to achieve the job of inserting into two tables simultaneously. The coding and creation of the view and trigger is given below

Example

```
Create view ord_view as select order_master. orderno, order_master . ostatus,
order_detail . qty_deld, order_detail . qty_ord from order_master, order_detail
where order_master . orderno = order_detail . orderno
/
```

When the above query is successfully compiled the output appears as shown below,

View created

Code to create a trigger on the above created view ord_view.

```
create or replace trigger order_mast_insert
INSTEAD OF insert on ord_view
referencing new as n each row
declare
cursor ecur is select * form order_master
where order_master . orderno = :n . orderno ;
cursor dcur is select * from order_detail
where order_detail . orderno = :n . orderno ;
a ecur%rowtype ;
b dcur%rowtype ;
begin
open ecur ;
open dcur ;
fetch ecur into a ;
fetch dcur into b ;
if dcur%notfound then
insert into order_master (orderno, o_status)
        values (:n . orderno, :n . o_status) ;
else
```

```
        update order_master set order_master . o_status = :n
        where order_master . orderno = :n . orderno ;
end if ;
if ecur%notfound then
        insert into order_detail (qty_ord, qty_deld, ordern)
        values (:n . qty_ord, :n . qty)deld, :n . orderno) ;
else
        update order_detail set order_detail . qty_ord = :n
        order_detail . qty_deld = :n . qty_deld
where order_detail . orderno = :n . orderno ;
end if ;
close ecur ;
close dcur ;
end ;
/
```

The output of the above code will be

Trigger created.

7.2.3 Creating Triggers

Creating DDL Event Trigger

As of Oracle 8i, you can create triggers that are executed when a DDL event occurs. If you are planning to use this feature solely for security purposes, you should investigate using the audit command instead. You can use a DDL event trigger to execute, function, index, package, procedure, role, sequence, synonym, table, type, or view. If you can use the on schema clause, the trigger will execute for any new data dictionary objects in your schema.

Example

```
create trigger CREATE_DB_OBJECT_AUDIT
after create on schema
begin
call INSERT_AUDIT_RECORDS (sys.dictionary_obj_name) ;
end ;
/
```

Creating Database Event Triggers

Like DML events, database events can execute triggers. When a database event occurs (a shutdown, startup, or error), you can execute a trigger that references the attributes of the event. You could a use database event to perform system maintenance functions immediately after each database startup. Pinning package is an effective way of keeping large PL/SQL objects in the shared pool of memory, improving performance and enhancing database stability. This trigger, PIN_ON_STARTUP, will run each time the database is started.

```
create or replace trigger PIN_ON_STARTUP
after startup on database
begin
DBMS_SHARED_POOL.KEEP (
        'SYS.STANDARD', 'P') ;
end ;
/
```

This example shows that trigger will be executed immediately after the database startup.

7.2.4 Modifying a Trigger

A trigger cannot be directly modified. To change the definition of a trigger you must recreate the trigger with the CREATE command. If a trigger had its privileges granted to other users, they remain valid as long as the trigger exists.

Enabling and Disabling Triggers

When a trigger is created it is automatically enabled and is triggered whenever the triggering command and the execution command is true. An enabled trigger executes the trigger body if the triggering statement is issued. To disable the execution of the use the ALTER TRIGGER command with the DISABLE clause. A disable trigger does not execute the trigger body even if the triggering statement is issued. We can disable / enable the trigger by the following syntax:

ALTER TRIGGER <trigger name> DISABLE / ENAMBLE

Calling Procedures Within Trigger

We can call procedures within a trigger to avoid writing large block of code in the trigger body. You can save the code of the stored procedure and call the procedure within the trigger, by using call command, as shown in the following syntax

```
Create or replace Trigger <trigger name>
[ before | after ] [ Insert  | Update | Delete] on < table name > [ for each statement / for each row ]
[ when < condition> ] ;
begin
call <procedure name>
(statements)
```

```
end ;
/
```

Deleting a Trigger

To delete a trigger use the DROP TRIGGER command

Syntax

DROP TRIGGER <trigger name> ;

This removes the trigger structure from the database and withdraws the privileges that were granted to other users.

Obtaining Information about a Trigger

Oracle has a view in data dictionary, accessible only by those with the privilege of DBA, that contains data from all the triggers created for the DATABASE in use. This table is called DBA_TRIGGERS. Following is a description of its contents (using System as login and manager as password to access this table).

Desc command is used in the following example.

```
SQL> desc DBA_TRIGGERS;
 Name                          Null?    Type
 ----------------------------- -------- ----
 OWNER                         NOT NULL VARCHAR2(30)
 TRIGGER_NAME                  NOT NULL VARCHAR2(30)
 TRIGGER_TYPE                           VARCHAR2(16)
 TRIGGERING_EVENT                       VARCHAR2(26)
 TABLE_OWNER                   NOT NULL VARCHAR2(30)
 TABLE_NAME                    NOT NULL VARCHAR2(30)
 REFERENCING_NAMES                      VARCHAR2(87)
 WHEN_CLAUSE                            VARCHAR2(4000)
 STATUS                                 VARCHAR2(8)
 DESCRIPTION                            VARCHAR2(4000)
 TRIGGER_BODY                           LONG
```

To view the user's triggers, you use the view USER_TRIGGERS:

```
SQL> desc user_triggers
 Name                          Null?    Type
 ----------------------------- -------- ----
 TRIGGER_NAME                  NOT NULL VARCHAR2 (30)
 TRIGGER_TYPE                           VARCHAR2 (16)
 TRIGGERING_EVENT                       VARCHAR2 (26)
```

```
Name                          Null?            Type
----------------------------  ---------------  ------------
 TABLE_OWNER                  NOT NULL         VARCHAR2 (30)
 TABLE_NAME                   NOT NULL         VARCHAR2 (30)
 REFERENCING_NAMES                             VARCHAR2 (87)
 WHEN_CLAUSE                                   VARCHAR2 (4000)
 STATUS                                        VARCHAR2 (8)
 DESCRIPTION                                   VARCHAR2 (4000)
 TRIGGER_BODY                                  LONG
```

Check Your Progress

Fill in the blanks:

1. Triggers may used to supplement declarative referential ..... , to enforce complex business rules, or to audit changes to data.
2. A trigger cannot execute the ..... , ROLLBACK, or SAVEPOINT commands.
3. A trigger's type is defined by the type of triggering ..... and by the level at which the trigger is executed.

7.3 LET US SUM UP

A trigger is automatically executed without any action required by the user. A stored procedure on other hand needs to be explicitly invoked. A trigger cannot execute the COMMIT, ROLLBACK, or SAVEPOINT commands. It also cannot call procedures or functions that execute those tasks. The SELECT command can be used only with the INTO clause. The trigger can be activated by a SQL command or by a user event. In a table, it can be triggered by the INSERT, UPDATE, or DELETE commands. A trigger cannot be directly modified. To change the definition of a trigger you must recreate the trigger with the CREATE command. If a trigger had its privileges granted to other users, they remain valid as long as the trigger exists.

7.4 KEYWORDS

Trigger Restriction: Restrictions on a trigger can be achieved using the WHEN clause.

Row Level Trigger: Row level triggers execute once for each row in a transaction.

Statement Level Trigger: Statement level triggers are triggered only once for each transaction.

Before and After Trigger: Since triggers are executed by events, they may be set to occur immediately before of after those events.

Instead of Trigger: These are triggers that are defined on a view rather than on a table.

Database-Level Trigger: You can create triggers to be fired on database events, including errors, logons, logoffs, shutdowns, and startups.

7.5 QUESTIONS FOR DISCUSSION

1. What is database triggers? In how many ways database triggers can be used?
2. Explain the difference between row level trigger and statement level trigger.
3. Discuss the creation of DDL event trigger and database event trigger.
4. How can we enable and disable the trigger?

Check Your Progress: Model Answers

1. Integrity
2. COMMIT
3. transaction

7.6 SUGGESTED READINGS

Peter Rob, Carlos Coronel, *Database Systems: Design, Implementation and Management*, Seventh edition, Thomson Learning, 2007

Silberschatz, Korth, Sudarshan, *Database System Concepts*, Fifth edition, McGraw-Hill, 2005

Elmasari Navathe, *Fundamentals of Database Systems*, Third edition, Pearson Education Asia, 2001

E. J. Yannakoudakis, *The Architectural Logic of Database Systems*, Springer-Verlag, Digitized 2007

Fred R. McFadden, Jeffrey A. Hoffer, *Database Management*, Benjamin/Cummings, Digitized 2007

Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems*, Third edition, McGraw-Hill Higher Education, 2003

LESSON

8

STORED PROCEDURES, FUNCTIONS AND PACKAGES

CONTENTS

8.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of stored procedures
- Discuss stored functions
- Describe the packages

8.1 INTRODUCTION

Procedures, also known as stored procedures, are stored in the database and are invoked or called by any anonymous block (The PL/SQL block that appears within an application).

In this lesson we will discuss the stored procedures and function. We will also discuss packages.

8.2 STORED PROCEDURES

Procedures are named PL/SQL blocks that can take parameters, perform an action and can be invoked. A procedure is generally used to perform an action and to pass values.

Procedures are made up of the following parts:

- A declarative part,
- An executable part,
- An optional exception-handling part.

Declarative Part: The declarative part may contain declarations of cursors, constants, variables, exceptions, and subprograms. These objects are local to the procedure. The objects become invalid once you exit from it.

Executable Part: The executable part contains a PL/SQL block consisting of statements that assign values, control execution and manipulate ORACLE data. The action to be performed is coded here and data that is to be returned back to the calling environment is also returned from here.

Exception Handling Part: This part contains code that performs an action to deal with exceptions raised during the execution of the Executable part. This block can be used to handle Oracle's own exceptions or the exceptions that are declared in the Declarative part. One cannot transfer the flow of execution from the Exception Handling part to the Executable part or vice-versa.

8.2.1 Where to Store Procedures?

Before the procedure is created, ORACLE parses the procedure. Then this parsed procedure is stored in the database.

Syntax for creating stored procedure:

```
CREATE OR REPLACE
PROCEDURE [schema.] procedurename
(argument { IN, OUT, IN OUT} datatype, ...) {IS, AS}
variable declarations;
constant declarations;
BEGIN
PL/SQL subprogram body;
EXCEPTION
exception PL/SQL block;
END;
```

8.2.2 How to Create and Execute Procedures?

When a procedure is created, ORACLE automatically performs the following steps:

1. Compiles the procedure.
2. Stores the compiled code.
3. Stores the procedure in the database.

The PL/SQL compiler compiles the code. If an error occurs, then the procedure is created but it is an invalid procedure. ORACLE displays a message during the time of creation that the procedure was created with compilation errors.

It does not display the errors. These errors can be viewed using the select statement.

SELECT * FROM *user_errors*;

ORACLE loads the compiled procedure in the memory area called the System Global Area (SGA). This allows the code to be executed quickly. The same procedure residing in the SGA is executed by the other users also.

Execution of Procedures

ORACLE performs the following steps to execute a procedure:

1. Verifies user access.
2. Verifies procedure validity.
3. Executes the procedure.

ORACLE checks if the user who called the procedure has the *execute* privilege for the procedure. If the user is invalid, then access is denied otherwise Oracle proceeds to check whether the called procedure is valid or not. The user can view the validity of the procedure by using the *select* statement as:

SELECT object_name, object_type, status FROM user____objects;

WHERE object_type = 'PROCEDURE';

Only if the status is *valid*, then the procedure can be executed. Once the procedure is found valid, ORACLE then loads the procedure into memory. (i.e. if it is not present in memory) and executes the PL/SQL code.

8.3 STORED FUNCTIONS

Functions are named PL/SQL blocks that can take parameters, perform an action and returns a value to the host environment. A function can only return one value.

Functions are made up of:

1. A declarative part,
2. An executable part,
3. An optional exception-handling part.

Declarative Part: The declarative part may contain declarations of type, cursors, constant, variables, exceptions, and subprograms. These objects are local to the function. The objects become invalid once you exit from the function. Here the datatype of the return value is also declared.

Executable Part: The executable part contains a PL/SQL block consisting of statements that assign values, control execution, and manipulate Oracle data. The action to be performed is coded here and data that is to be returned back to the calling environment is also returned from here. Variable declared is put to use in this block. The return value is also passed back in this part.

Exception Handling Part: This part contains code that performs an action to deal with exceptions raised during execution of the Executable Part. This block can be used to handle Oracle's own exceptions or the exceptions that are declared in the Declarative Part. One cannot transfer the flow of execution from the Exception Handling Part to the Executable Part and vice-versa. The return value can also be passed back in this part.

8.3.1 Where to Store Functions ?

Functions in Oracle are called stored functions. Functions are stored in the database and are invoked or called by any anonymous block (a PL/SQL block that appears within an application) Before the function is created, Oracle parses the function. Then this parsed function is stored in the database.

8.3.2 Create and Execute Functions

When a function is created, Oracle automatically performs the following steps:

1. Compiles the function.
2. Stores the compiled code.
3. Stores the function in the database.

The PL/SQL compiler compiles the code. If an error occurs then the function is created but it's an invalid function. Oracle displays a message during the time of creation that the function was created with compilation errors. It does not display the errors. These errors can be viewed by using the select statement:

SELECT * FROM *user_errors*;

Oracle loads the compiled function in the memory area called the System Global Area (SGA). This allows the code to be executed quickly. The same function residing in the SGA is executed by the other users also.

Syntax for creating a stored function:

```
CREATE OR REPLACE
        FUNCTION [schema.] functionname (argument IN datatype, ...)
                RETURN datatype {IS, AS}
                        variable declarations;
                        constant declarations;
                BEGIN
```

```
        PL/SQL subprogram body;
EXCEPTION
        exception PL/SQL block;
END;
```

Executing a Function

Oracle performs the following steps to execute a function:

1. Verifies user access.
2. Verifies function validity.
3. Executes the function.

Oracle checks if the user who called the function has the execute privilege for the function. If the user is invalid, then access is denied else if the user is valid, then it proceeds to check whether the called function is valid or not. The user can view the validity of the function by using the select statement as:

```
SELECT object_name, object_type, status
FROM user_objects,
WHERE object_type = 'FUNCTION';
```

Only if the status is valid, the function can be executed. Once the function is found valid, Oracle loads the function into memory (i.e. if it is not currently present in memory) and executes the PL/SQL code.

8.3.3 Advantages of Functions

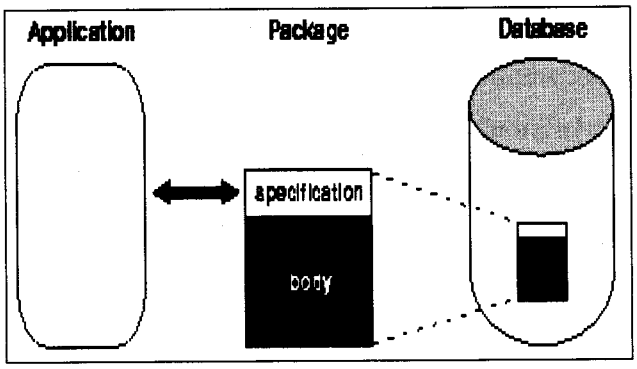
1. ***Security:*** Stored functions can help enforce data security. For example you can grant users access to function that can query a table, but not grant them access to the table itself.
2. ***Performance:*** It improves database performance in the following ways:
   - Amount of information sent over a network is less. No compilation step is required to execute the code.
   - As function is present in the shared pool of SGA, retrieval from disk is not required.
3. ***Memory Allocation:*** Reduction in memory as stored functions have shared memory capabilities so only one copy of function needs to be loaded for execution by multiple users.
4. ***Productivity:*** Increased development productivity, by writing a single function we can avoid redundant coding and increase productivity.
5. ***Integrity:*** Improves integrity, a function needs to be tested only once to guarantee that it returns an accurate result. So committing coding errors can be reduced.

8.4 PACKAGES

A **package** is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. The **specification** (**spec** for short) is the interface to your applications; it declares the

types, variables, constants, exceptions, cursors, and subprograms available for use. The **body** fully defines cursors and subprograms, and so implements the spec.

As the following figure shows, you can think of the spec as an operational interface and of the body as a "black box." You can debug, enhance, or replace a package body without changing the interface (package spec) to the package.



To create packages, use the CREATE PACKAGE statement, which you can execute interactively from SQL*Plus. Here is the syntax:

```
CREATE [OR REPLACE] PACKAGE package_name
        [AUTHID {CURRENT_USER | DEFINER}]
        {IS | AS}
        [PRAGMA SERIALLY_REUSABLE;]
        [collection_type_definition ...]
        [record_type_definition ...]
        [subtype_definition ...]
        [collection_declaration ...]
        [constant_declaration ...]
        [exception_declaration ...]
        [object_declaration ...]
        [record_declaration ...]
        [variable_declaration ...]
        [cursor_spec ...]
        [function_spec ...]
        [procedure_spec ...]
        [call_spec ...]
        [PRAGMA RESTRICT_REFERENCES(assertions) ...]
END [package_name];
```

```
[CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}
[PRAGMA SERIALLY_REUSABLE;]
[collection_type_definition ...]
[record_type_definition ...]
[subtype_definition ...]
[collection_declaration ...]
[constant_declaration ...]
[exception_declaration ...]
[object_declaration ...]
[record_declaration ...]
[variable_declaration ...]
[cursor_body ...]
[function_spec ...]
[procedure_spec ...]
[call_spec ...]
BEGIN
        sequence_of_statements]
END [package_name];]
```

The spec holds **public declarations**, which are visible to your application. You must declare subprograms at the end of the spec after all other items.

The body holds implementation details and **private declarations**, which are hidden from your application. Following the declarative part of the package body is the optional initialization part, which typically holds statements that initialize package variables.

The AUTHID clause determines whether all the packaged subprograms execute with the privileges of their definer (the default) or invoker, and whether their unqualified references to schema objects are resolved in the schema of the definer or invoker.

A **call spec** lets you publish a Java method or external C function in the Oracle data dictionary. The call spec publishes the routine by mapping its name, parameter types, and return type to their SQL counterparts.

In the example below, you package a record type, a cursor, and two employment procedures. Notice that the procedure hire_employee uses the database sequence empno_seq and the function SYSDATE to insert a new employee number and hire date, respectively.

```
CREATE OR REPLACE PACKAGE emp_actions AS  — spec
        TYPE EmpRecTyp IS RECORD (emp_id INT, salary REAL);
        CURSOR desc_salary RETURN EmpRecTyp;
```

```
        PROCEDURE hire_employee (
                ename  VARCHAR2,
                job    VARCHAR2,
                mgr    NUMBER,
                sal    NUMBER,
                comm   NUMBER,
                deptno NUMBER);
        PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;
CREATE OR REPLACE PACKAGE BODY emp_actions AS  — body
        CURSOR desc_salary RETURN EmpRecTyp IS
                SELECT empno, sal FROM emp ORDER BY sal DESC;
        PROCEDURE hire_employee (
                ename  VARCHAR2,
                job    VARCHAR2,
                mgr    NUMBER,
                sal    NUMBER,
                comm   NUMBER,
                deptno NUMBER) IS
                BEGIN
        INSERT INTO emp VALUES (empno_seq.NEXTVAL, ename, job,
                        mgr, SYSDATE, sal, comm, deptno);
        END hire_employee;
        PROCEDURE fire_employee (emp_id NUMBER) IS
        BEGIN
                DELETE FROM emp WHERE empno = emp_id;
        END fire_employee;
END emp_actions;
```

Only the declarations in the package spec are visible and accessible to applications. Implementation details in the package body are hidden and inaccessible. So, you can change the body (implementation) without having to recompile calling programs.

8.4.1 Advantages of PL/SQL Packages

Packages Offer Several Advantages: modularity, easier application design, information hiding, added functionality, and better performance.

Modularity: Packages let you encapsulate logically related types, items, and subprograms in a named PL/SQL module. Each package is easy to understand, and the interfaces between packages are simple, clear, and well defined. This aids application development.

Easier Application Design: When designing an application, all you need initially is the interface information in the package specs. You can code and compile a spec without its body. Then, stored subprograms that reference the package can be compiled as well. You need not define the package bodies fully until you are ready to complete the application.

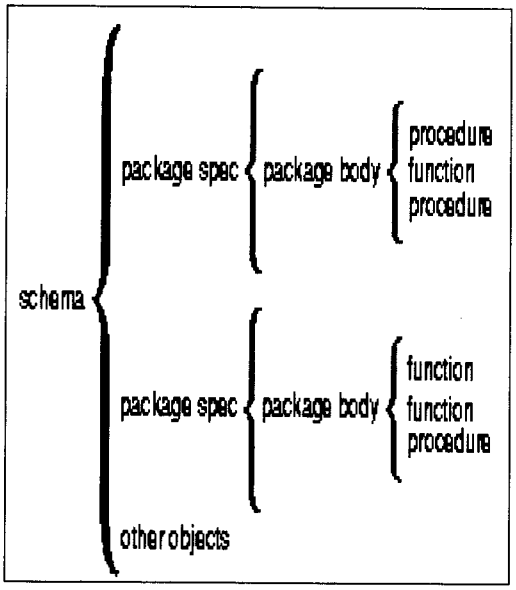
Information Hiding: With packages, you can specify which types, items, and subprograms are public (visible and accessible) or private (hidden and inaccessible). For example, if a package contains four subprograms, three might be public and one private. The package hides the implementation of the private subprogram so that only the package (not your application) is affected if the implementation changes. This simplifies maintenance and enhancement. Also, by hiding implementation details from users, you protect the integrity of the package.

Added Functionality: Packaged public variables and cursors persist for the duration of a session. So, they can be shared by all subprograms that execute in the environment. Also, they allow you to maintain data across transactions without having to store it in the database.

Better Performance: When you call a packaged subprogram for the first time, the whole package is loaded into memory. So, later calls to related subprograms in the package require no disk I/O. Also, packages stop cascading dependencies and thereby avoid unnecessary recompiling. For example, if you change the implementation of a packaged function, Oracle need not recompile the calling subprograms because they do not depend on the package body.

8.4.2 Understanding the Package Spec

The package spec contains public declarations. The scope of these declarations is local to your database schema and global to the package. So, the declared items are accessible from your application and from anywhere in the package as shown below:



The spec lists the package resources available to applications. All the information your application needs to use the resources is in the spec. For example, the following declaration shows that the function named fac takes one argument of type INTEGER and returns a value of type INTEGER:

```
FUNCTION fac (n INTEGER) RETURN INTEGER; — returns n!
```

That is all the information you need to call the function. You need not consider its underlying implementation (whether it is iterative or recursive for example). Only subprograms and cursors have an underlying implementation. So, if a spec declares only types, constants, variables, exceptions, and call specs, the package body is unnecessary. Consider the following bodiless package:

```
CREATE PACKAGE trans_data AS  — bodiless package
        TYPE TimeRec IS RECORD (
                minutes SMALLINT,
                hours   SMALLINT);
        TYPE TransRec IS RECORD (
                category VARCHAR2,
                account  INT,
                amount   REAL,
time_of  TimeRec);
                minimum_balance   CONSTANT REAL := 10.00;
                number_processed  INT;
                insufficient_funds EXCEPTION;
END trans_data;
```

The package trans_data needs no body because types, constants, variables, and exceptions do not have an underlying implementation. Such packages let you define global variables—usable by subprograms and database triggers—that persist throughout a session.

8.4.3 Referencing Package Contents

To reference the types, items, subprograms, and call specs declared within a package spec, use dot notation, as follows:

```
package_name.type_name
package_name.item_name
package_name.subprogram_name
package_name.call_spec_name
```

You can reference package contents from database triggers, stored subprograms, 3GL application programs, and various Oracle tools. For example, you might call the packaged procedure hire_employee from SQL*Plus, as follows:

```
SQL> CALL emp_actions.hire_employee('TATE', 'CLERK', ...);
```

In the example below, you call the same procedure from an anonymous PL/SQL block embedded in a Pro*C program. The actual parameters emp_name and job_title are host variables (that is, variables declared in a host environment).

```
EXEC SQL EXECUTE
BEGIN
        emp_actions.hire_employee(:emp_name, :job_title, ...);
```

You cannot reference remote packaged variables directly or indirectly. For example, you cannot call the following procedure remotely because it references a packaged variable in a parameter initialization clause:

```
CREATE PACKAGE random AS
    seed NUMBER;
    PROCEDURE initialize (starter IN NUMBER := seed, ...);
```

Also, inside a package, you cannot reference host variables.

8.4.4 The Package Body

The package body implements the package spec. That is, the package body contains the implementation of every cursor and subprogram declared in the package spec. Keep in mind that subprograms defined in a package body are accessible outside the package only if their specs also appear in the package spec.

To match subprogram specs and bodies, PL/SQL does a token-by-token comparison of their headers. So, except for white space, the headers must match word for word. Otherwise, PL/SQL raises an exception, as the following example shows:

```
CREATE PACKAGE emp_actions AS
  ...
  PROCEDURE calc_bonus (date_hired emp.hiredate%TYPE, ...);
END emp_actions;
CREATE PACKAGE BODY emp_actions AS
  ...
  PROCEDURE calc_bonus (date_hired DATE, ...) IS
    -- parameter declaration raises an exception because 'DATE'
    -- does not match 'emp.hiredate%TYPE' word for word
  BEGIN ... END;
END emp_actions;
```

The package body can also contain private declarations, which define types and items necessary for the internal workings of the package. The scope of these declarations is local to the package body. Therefore, the declared types and items are inaccessible except from within the package body. Unlike a package spec, the declarative part of a package body can contain subprogram bodies.

Following the declarative part of a package body is the optional initialization part, which typically holds statements that initialize some of the variables previously declared in the package.

The initialization part of a package plays a minor role because, unlike subprograms, a package cannot be called or passed parameters. As a result, the initialization part of a package is run only once, the first time you reference the package.

Remember, if a package spec declares only types, constants, variables, exceptions, and call specs, the package body is unnecessary. However, the body can still be used to initialize items declared in the package spec.

Consider the package below named emp_actions. The package spec declares the following types, items, and subprograms:

- Types EmpRecTyp and DeptRecTyp
- Cursor desc_salary
- Exception invalid_salary
- Functions hire_employee and nth_highest_salary
- Procedures fire_employee and raise_salary

After writing the package, you can develop applications that reference its types, call its subprograms, use its cursor, and raise its exception. When you create the package, it is stored in an Oracle database for general use.

```
CREATE PACKAGE emp_actions AS
        TYPE EmpRecTyp IS RECORD (emp_id INT, salary REAL);
        TYPE DeptRecTyp IS RECORD (dept_id INT, location VARCHAR2);
        CURSOR desc_salary RETURN EmpRecTyp;
        invalid_salary EXCEPTION;
        FUNCTION hire_employee (ename  VARCHAR2, job    VARCHAR2, mgr    REAL,
        sal    REAL, comm   REAL, deptno REAL) RETURN INT;
        PROCEDURE fire_employee (emp_id INT);
        PROCEDURE raise_salary (emp_id INT, grade INT, amount REAL);
        FUNCTION nth_highest_salary (n INT) RETURN EmpRecTyp;
        END emp_actions;
CREATE PACKAGE BODY emp_actions AS number_hired INT;  - visible only in this package
        CURSOR desc_salary RETURN EmpRecTyp IS
        SELECT empno, sal FROM emp ORDER BY sal DESC;
        FUNCTION hire_employee (ename  VARCHAR2, job VARCHAR2, mgr    REAL,
        sal    REAL, comm   REAL, deptno REAL) RETURN INT IS new_empno INT;
        BEGIN
        SELECT empno_seq.NEXTVAL INTO new_empno FROM dual;
INSERT INTO emp VALUES (new_empno, ename, job, mgr, SYSDATE, sal, comm, deptno);
      number_hired := number_hired + 1;
```

```
        RETURN new_empno;
                END hire_employee;
                PROCEDURE fire_employee (emp_id INT) IS
                BEGIN
        DELETE FROM emp WHERE empno = emp_id;
                END fire_employee;
FUNCTION sal_ok (rank INT, salary REAL) RETURN BOOLEAN IS min_sal REAL;
                max_sal REAL;
                BEGIN
SELECT losal, hisal INTO min_sal, max_sal FROM salgrade WHERE grade = rank;
RETURN (salary >= min_sal) AND (salary <= max_sal);
                END sal_ok;
PROCEDURE raise_salary (emp_id INT, grade INT, amount REAL) IS salary REAL;
                BEGIN
        SELECT sal INTO salary FROM emp WHERE empno = emp_id;
        IF sal_ok(grade, salary + amount) THEN
                UPDATE emp SET sal = sal + amount WHERE empno = emp_id;
        ELSE
                RAISE invalid_salary;
        END IF;
                END raise_salary;
 FUNCTION nth_highest_salary (n INT) RETURN EmpRecTyp IS emp_rec EmpRecTyp;
                BEGIN
                        OPEN desc_salary;
                        FOR i IN 1..n LOOP
                                        FETCH desc_salary INTO emp_rec;
                        END LOOP;
                        CLOSE desc_salary;
                        RETURN emp_rec;
                END nth_highest_salary;
        BEGIN
                INSERT INTO emp_audit VALUES (SYSDATE, USER, 'EMP_ACTIONS');
                number_hired := 0;
        END emp_actions;
```

Remember, the initialization part of a package is run just once, the first time you reference the package. So, in the last example, only one row is inserted into the database table emp_audit. Likewise, the variable number_hired is initialized only once. Every time the procedure hire_employee is called, the variable number_hired is updated. However, the count kept by

number_hired is session specific. That is, the count reflects the number of new employees processed by one user, *not* the number processed by all users.

8.4.5 Overview of Product-Specific Packages

Oracle and various Oracle tools are supplied with product-specific packages that help you build PL/SQL-based applications. For example, Oracle is supplied with many utility packages, a few of which are highlighted below.

DBMS_ALERT Package: Package DBMS_ALERT lets you use database triggers to alert an application when specific database values change. The alerts are transaction based and asynchronous (that is, they operate independently of any timing mechanism). For example, a company might use this package to update the value of its investment portfolio as new stock and bond quotes arrive.

DBMS_OUTPUT Package: Package DBMS_OUTPUT enables you to display output from PL/SQL blocks and subprograms, which makes it easier to test and debug them. The procedure put_line outputs information to a buffer in the SGA. You display the information by calling the procedure get_line or by setting SERVEROUTPUT ON in SQL*Plus. For example, suppose you create the following stored procedure:

```
CREATE PROCEDURE calc_payroll (payroll OUT NUMBER) AS
      CURSOR c1 IS SELECT sal, comm FROM emp;
BEGIN
      payroll := 0;
      FOR c1rec IN c1 LOOP
            c1rec.comm := NVL(c1rec.comm, 0);
            payroll := payroll + c1rec.sal + c1rec.comm;
      END LOOP;
      dbms_output.put_line('Value of payroll: ' || TO_CHAR(payroll));
END;
```

When you issue the following commands, SQL*Plus displays the value assigned by the procedure to parameter payroll:

```
SQL> SET SERVEROUTPUT ON
SQL> VARIABLE num NUMBER
SQL> CALL calc_payroll(:num);
Value of payroll: 31225
```

DBMS_PIPE Package: Package DBMS_PIPE allows different sessions to communicate over named pipes. (A *pipe* is an area of memory used by one process to pass information to another.) You can use the procedures pack_message and send_message to pack a message into a pipe then send it to another session in the same instance.

At the other end of the pipe, you can use the procedures receive_message and unpack_message to receive and unpack (read) the message. Named pipes are useful in many ways. For example, you can write routines in C that allow external programs to collect information, then send it through pipes to procedures stored in an Oracle database.

***UTL_FILE Package*:** Package UTL_FILE allows your PL/SQL programs to read and write operating system (OS) text files. It provides a restricted version of standard OS stream file I/O, including open, put, get, and close operations.

When you want to read or write a text file, you call the function fopen, which returns a file handle for use in subsequent procedure calls. For example, the procedure put_line writes a text string and line terminator to an open file, and the procedure get_line reads a line of text from an open file into an output buffer.

***UTL_HTTP Package*:** Package UTL_HTTP allows your PL/SQL programs to make HyperText Transfer Protocol (HTTP) callouts. It can retrieve data from the Internet or call Oracle Web Server cartridges. The package has two entry points, each of which accepts a URL (uniform resource locator) string, contacts the specified site, and returns the requested data, which is usually in HyperText Markup Language (HTML) format.

8.4.6 Guidelines for Writing Packages

When writing packages, keep them as general as possible so they can be reused in future applications. Avoid writing packages that duplicate some feature already provided by Oracle. Package specs reflect the design of your application. So, define them before the package bodies. Place in a spec only the types, items, and subprograms that must be visible to users of the package. That way, other developers cannot misuse the package by basing their code on irrelevant implementation details.

To reduce the need for recompiling when code is changed, place as few items as possible in a package spec. Changes to a package body do not require Oracle to recompile dependent procedures. However, changes to a package spec require Oracle to recompile every stored subprogram that references the package.

Check Your Progress

Fill in the blanks:

1. The ..... part may contain declarations of cursors, constants, variables, exceptions, and subprograms.
2. Functions are stored in the database and are invoked or called by any .....
3. Packages usually have two parts, a ..... and a body, although sometimes the body is unnecessary.
4. Package ..... lets you use database triggers to alert an application when specific database values change.

8.5 LET US SUM UP

Procedures are named PL/SQL blocks that can take parameters, perform an action and can be invoked. A procedure is generally used to perform an action and to pass values. Functions are named PL/SQL blocks that can take parameters, perform an action and returns a value to the host environment. A function can only return one value. A **package** is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. Oracle and various Oracle tools are supplied with product-specific packages that help you build PL/SQL-based applications.

8.6 KEYWORDS

Procedure: Procedures, also known as stored procedures, are stored in the database and are invoked or called by any anonymous block.

Stored Functions: Functions are named PL/SQL blocks that can take parameters, perform an action and returns a value to the host environment.

Package: A package is a schema object that groups logically related PL/SQL types, items, and subprograms.

DBMS_PIPE Package: Package DBMS_PIPE allows different sessions to communicate over named pipes.

8.7 QUESTIONS FOR DISCUSSION

1. Explain the stored procedures. Write the syntax to store the procedures.
2. Discuss the various parts of stored functions and stored procedures.
3. How to create and execute the function?
4. What are the advantages of stored functions?
5. Discuss the packages in PL/SQL. Explain with its advantages.
6. Explain the term "The Package Spec".

Check Your Progress: Model Answers

1. Declarative
2. anonymous block
3. specification
4. DBMS_ALERT

8.8 SUGGESTED READINGS

Peter Rob, Carlos Coronel, *Database Systems: Design, Implementation and Management*, Seventh edition, Thomson Learning, 2007

Silberschatz, Korth, Sudarshan, *Database System Concepts*, Fifth edition, McGraw-Hill, 2005

Elmasari Navathe, *Fundamentals of Database Systems*, Third edition, Pearson Education Asia, 2001

E. J. Yannakoudakis, *The Architectural Logic of Database Systems*, Springer-Verlag, Digitized 2007

Fred R. McFadden, Jeffrey A. Hoffer, *Database Management*, Benjamin/Cummings, Digitized 2007

Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems*, Third edition, McGraw-Hill Higher Education, 2003

UNIT V

LESSON

9

DISTRIBUTED PROCESSING

CONTENTS

9.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the concept of distributed database
- Discuss data fragmentation
- Describe the significance of data replication
- Identify and explain the data allocation
- Discuss the query processing in distributed database

9.1 INTRODUCTION

The databases described in the last lessons were essentially general purpose databases which could be tailored and customized to suit a given data management and processing situation. However, there are a number of other applications where special features other than general database functions are required. This lesson deals with a special database management system where the components of the system are physically located at different places.

9.2 DISTRIBUTED DATABASE

A database that physically resides entirely on one machine under a single DBMS is known as local database management system. The database management system that resides entirely on a machine different from that of the user connected through a network is known as remote database. In either case the entire database is controlled by a single site and hence is knows as Centralized Database System. In contrast to this a database may be fragmented and each of its fragments is stored on different machines connected through network(s) or is controlled by different DBMSs or operates under different operating systems. Such a multiple-source and multiple-location database is called distributed database.

More formally, a Distributed database (or a DDB) is a collection of multiple logically interrelated databases distributed over a computer network. A distributed database management system (DDBMS) is a software system that manages a distributed database while making the distribution transparent to the user. The user is unaware that the database is fragmented. The Distributed Database Management System ensures that the users access the distributed database as if it were a local database. A collection of files stored at different nodes of network and maintaining of Inter relationship among them via hyperlinks has become a common organization on the Internet, with file of web pages.

Typically, a distributed database system consists of a collection of sites, each of which maintains a local database system (Figure 9.1). Each site is able to process local transactions, those transactions that access data only in that single site. In addition, a site may participate in the execution of global transactions, those transactions that access data is several sites. The execution of global transactions requires communication among the sites usually through a network.

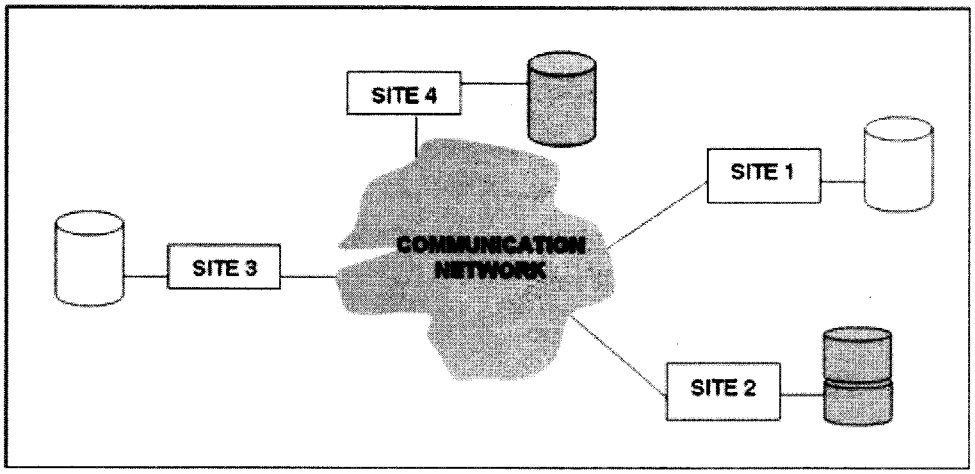


Figure 9.1: Distributed Database Architecture

The sites in the system can be connected physically in a variety of ways. The various topologies are represented as graphs whose nodes correspond to sites. An edge from node A to node B corresponds to a direct connection between the two sites.

Exactly how a database is distributed is known as its configuration and they differ from each other in the following aspects:

- ***Installation Cost***: The cost of physically linking the sites in the system.
- ***Communication Cost***: The cost in time and money to send a message from site A to site B.
- ***Reliability***: The frequency with which a link or site fails.
- ***Availability***: The degree to which data can be accessed despite the failure of some links or sites.

These differences play an important role in choosing the appropriate mechanism for handling the distribution of data.

The participating or collaborating sites of a distributed database system may be distributed physically either over a large geographical area such as the all-Indian state capitals or over a small geographical area such as a single building or a number of adjacent building. The former type of network is referred to as a long-haul network or wide area network, while the latter is referred to as a local-area network.

Since the sites in long-haul network are distributed physically over a large geographical area, the communication links are likely to be relatively slow and less reliable as compared with local area networks. Typical long-haul links are telephone lines, microwave links, and satellite channels.

In contrast, since all the sites in local-area networks are close to each other communication links are of higher speed and lower error rate than their counterparts in long-haul networks. The most common channels are twisted pair, base band coaxial, broadband coaxial, and fiber optics.

The links of a network between its nodes may be of different patterns knows as its topology. Some of the network topologies are depicted in Figure 9.2.

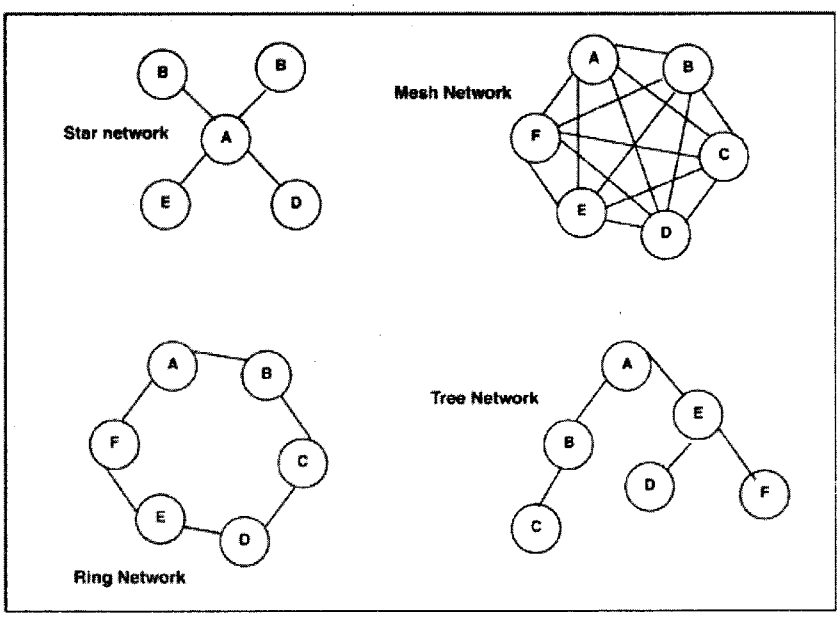


Figure 9.2: Network Topologies

9.2.1 Data Distribution Advantages

Distributed database systems have a number of advantages over their centralized counterparts. The primary goal of distributed database systems is to achieve the ability to share and access data stored in databases spread across different machines, operating systems and DBMSs, in a reliable, fast and efficient manner. The benefits of distributed database are explained below.

- ***Space independence:*** If a number of different sites are connected to each other, then a user at one site may be able to access data that is available at another site. The user does not have to be present physically at the database sit. Therefore, the database becomes space independent. Thus, through distributed database system, a user can access the database physically stored at University of Delhi in Delhi without being at the site.
- ***Availability of data where it is needed:*** The data in a distributed database system are so dispersed as to match the data requirements of the users.
- ***Faster data access:*** The end-users only with a subset of the entire database. If this portion of the database is locally stored and accessed, it will be many times faster than when remotely located.
- ***Faster Data Processing:*** For same reason as above the data processing the users' end will be considerably faster.
- ***Distributed control:*** The primary benefit to accomplishing data sharing by means of data distribution is that each site is able to retain a degree of control over data stored locally. In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system.
- ***User-friendly interface:*** The end users are free to have interfaces of their own choice at their sites.
- ***Increased Reliability:*** In case of a centralized database system, a failure renders the entire system useless. Such is not the case with the distributed database systems. Even in case of a failure the end users still can access their own database stored locally.
- ***Query speedup:*** If a query involves data at several sites, it may be possible to split the query into sub-queries that can be excited in parallel by several sites. Such parallel computation allows faster processing of a user's query. In those cases in which data is replicated, queries may be directed by the system to the least heavily loaded sites.

9.2.2 Data Distribution Disadvantages

Distributed database systems are not entirely free from limitations. The primary drawback of distributed database systems is the added complexity required to ensure proper coordination among the sites. This increased complexity takes the form of:

- ***Complexity of management and control:*** All the related management activities and control of the same becomes very complex with degree of distribution.
- ***Software development cost:*** It is more difficult to implement a distributed database system and, thus, is more costly as compared to centralized local database.
- ***Higher possibility of bugs:*** Since the sites that comprise the distributed system operate in parallel, it is harder to ensure the correctness of algorithms. This mode of operation makes them extremely vulnerable to bugs. The art of constructing distributed algorithms remains an active and important area or research.

- ***Increased processing overhead:*** The exchange of data, messages and the additional computation required to achieve inter-site coordination is a form of overhead that does not arise in centralized systems. For a single transaction the overhead is more than ten times in general.
- ***Lack of standards:*** Every user of a distributed database system is free to have her own standard and no common protocol may exist.
- ***Security:*** Because of its and extent, distributed database systems are vulnerable to security lapses. Network communication being an integral part of such systems, security concerns are more frequent than centralized database systems.

As is evident from above discussion that distributed database systems have both specific advantages and disadvantages. An optimal trade-off between the distribution and centralization may be employed to arrive at the right kind of design. Therefore, in choosing the design for a database system, the designer must balance the advantages against the disadvantages of distribution of data design ranging from fully distributed designs to designs which include large degree of centralization.

9.2.3 Functions of Distributed Database Management System

Distribution leads to increased complexity in system design and implementation. To achieve the potential advantages of DDBMS as listed earlier; the DDBMS software must be able to provide the following functions in addition to those of a centralized DBMS.

1. ***Keeping track of data:*** The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.
2. ***Distributed query processing:*** The ability to access remote sites and transmit queries and data among the various sites via a communication network.
3. ***Distributed transaction management:*** The ability to devise execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain integrity of the overall database.
4. ***Replicated data management:*** The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of replicated data items.
5. ***Distributed database recovery:*** The ability to recover from individual crashes and from new types of failures such as the failure of a communication links.
6. ***Security:*** Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.
7. ***Distributed directory (catalog) management:*** A directory contains information (meta data) about data in the database. The directory may be global for the entire DDB or local for each site.

9.2.4 Components of Distributed Database Management System

A DDBMS has many components connected together. Some of the components that a DDBMS must have are:

- ***Sites or Nodes (Workstations):*** The end users machines (mostly PCs) that form the network. The distributed database system is independent of the hardware of the workstations.

- ***Network hardware and software:*** Each workstation must have necessary hardware and software that enable them to establish a network with other components on the distributed database system. The DDBase system should be independent of the network type of each workstation.
- ***Transaction processor (TP):*** Each of the data-requesting workstation must have this software component that receives and processes the request for data (local or remote). It makes the data access transparent to the user. TP is also sometimes called application processor (AP) or transaction manager TM.
- ***Data processor (DP):*** It is a software component on each participating computer in the distributed database system. This component stores and retrieves data located at that particular site. It is also known as data manager (DM). A centralized DBMS may also act as a DM on a site.

9.2.5 Levels of Data and Process Distribution

In a multiple site configuration, the responsibilities of each site may be different from each other. In a Distributed processing system the data comes from a centralized database systems but the processing is performed on more than one sites. For instance a computer system does all the data entry at Hisar, a computer located at Chandigarh does the validation checks on data whereas the statistical analysis is performed by a computer situated at Delhi. The data is actually stored in a DBMS located at Mumbai. This is an instance of distributed processing as depicted in Figure 9.3.

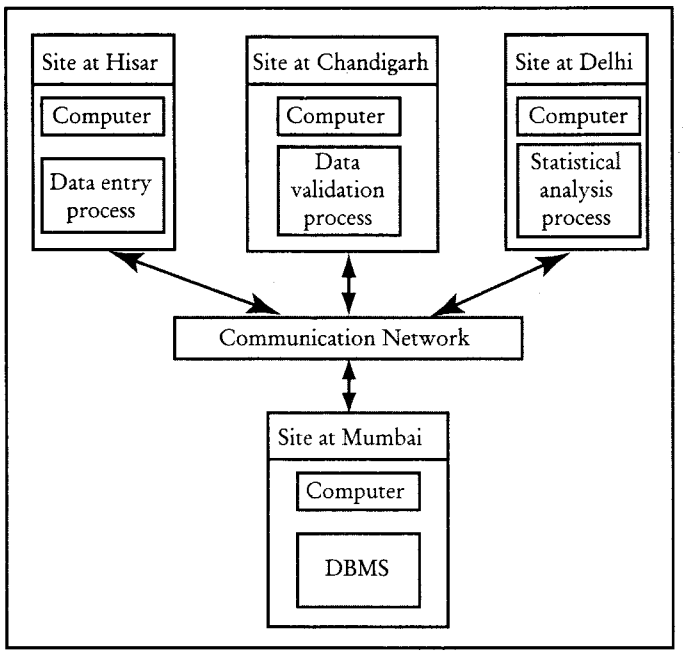


Figure 9.3: Distributed Processing Environment

In contrast to the above distributed processing environment, a distributed database system has the database split into fragments which may be physically located at different sites. Thus, the database may be stored at sites at Hisar, Chandigarh and Delhi while a process at Mumbai access these database fragments from Mumbai as shown in the Figure 9.4.

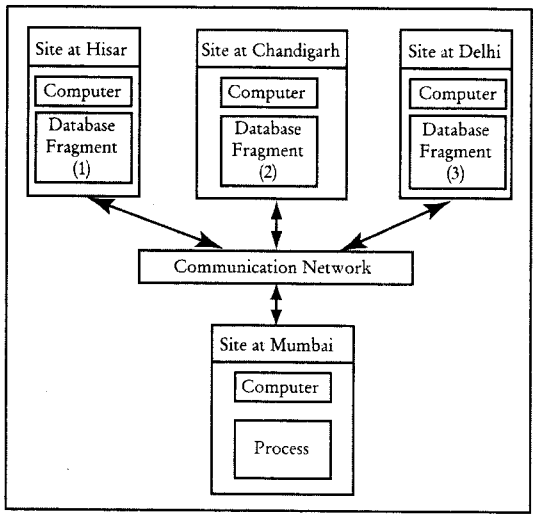


Figure 9.4: Distributed Database Environment

From the above discussion it is clear that:

- Distributed processing does not require a distributed database system, but a distributed database system does require a distributed processing system.
- A network support is required in both distributed database and processing systems.

9.2.6 Types of Distributed Database Systems

According to the way of database system and processing system are distributed as described above, a number of distributed configuration is possible. Some of the well known configurations are discussed below.

- ***Single-site processing single-site data (SPSD)***: In this configuration all processing is done on a single CPU or host computer usually a mainframe or mini-computer and all the data are stored on the local disks of this computer. The DBMS is located at a particular computer and is accesses by dumb terminals attached to it through network.
- ***Multiple-site processing single-site data (MPSD)***: In this configuration processes run on different sites accessing and sharing common database. A client-server configuration is one such example. The server is the computer providing the requested data by clients.
- ***Client-server configuration***: Sharing the computing abilities of different machines motivated the development of Client-Server architecture. A client is a component (hardware or software) that initiates a request for a service provided by another component called server. The server, in turn, processes the request, generates the requested result and passes the result back to the client.
- ***Multiple-site processing multiple-site data (MPMD)***: In this configuration, both the processes and database system are located at different sites. This is a scenario of fully-distributed system. MPMD may be further classified into:
  - ❖ Homogeneous distributed database systems, in which all the constituting databases are of same type.
  - ❖ Heterogeneous distributed database systems, in which the constituting databases are of different types. They may be thus relational, hierarchical, network or combination of these.

9.3 DATA FRAGMENTATION

The principles outlined in earlier unit in designing a centralized database are applicable even in the case of distributed database. However, there are few additional issues that arise in case of distributed database designing. They are:

- Data fragmentation
- Data replication
- Data allocation

The information concerning data fragmentation, replication and allocation is stored in a global directory that is accessed by the DDBS applications as needed.

It is clear that in a distributed database system the database is broken into smaller pieces. Here we will discuss the techniques that are used to break up the database into logical units, called fragments, which may be assigned for storage at the various sites.

If a relation R is fragmented, R is divided into a number of fragment relations R_1, R_2....., R_n. These fragments contain sufficient information to reconstruct the original relation R. This reconstruction can take place through the application of either the union operation or a special type of join operation on the various fragments depending on how they were obtained from the original relation. Of many methods of fragmentation, two of them shall be discussed here: horizontal fragmentation and vertical fragmentation.

Horizontal fragmentation splits the relation by assigning each tuple of R to one or more fragments. Vertical fragmentation splits the relation by decomposing the scheme R of relation R in a special way that we shall discuss. These two schemes can be applied successively to the same relation, resulting in a number of different fragments. Note that some information may appear in several fragments.

For illustration purposes, let us consider the customer relation CUSTOMER of some company:

CUSTOMER (CUS_ID, CUS_NAME, CUS_STATE, CUS_ DEPOSIT, CUS_BALANCE, CUS_RATING, CUS_DUE)

A sample instance of the CUSTOMER relation is shown below:

CUSTOMER	CUS_ ID	CUS_ NAME	CUS_ SATE	CUS_ DEPOSIT	CUS_ BALANCE	CUS_ RATING	CUS_ DUE
	10	Puranchand	Haryana	3000	2000	3	1000
	11	Rohit	Punjab	4000	3000	2	1500
	21	Ramlal	Haryana	2000	190	3	280
	23	Pankaj	Bihar	2300	230	3	320
	33	Rahul	Punjab	3300	450	2	400
	43	Satbir	Haryana	4500	1000	1	900

9.3.1 Horizontal Fragmentation

Under this fragmentation scheme, a able (or relation) r is partitioned into a number of subsets, r_1, r_2,..... Each subset r_i ($i=1,2...$) consists of a number of tuples of relation r. Each tuple of relation r must belong to one of the fragments, so that the original relation can be reconstructed whenever needed.

A fragment may be defined as a selection on the global relation r. That is, the union of all the fragments should be able to generate the original relation.

In our sample relation CUSTOMER, assume that each state headquarters requires data belonging to that state only. Therefore, the relation can be horizontally fragmented as given:

Fragment Name	Location of fragment	Selection condition	Node name	Customer_IDs	Number of rows
CUS_BHR	Patna	CUS_SATE="Bihar"	BHS	23	1
CUS_HAR	Hisar	CUS_SATE="Haryana"	HRS	10, 21, 43	3
CUS_PUN	Amritsar	CUS_SATE="Punjab"	PNS	11, 33	2

The three resulting fragment relations are:

Fragment Name : CUS_BHR		Location: Patna		Node:BHS		
CUS_ID	CUS_NAME	CUS_SATE	CUS_DEPOSIT	CUS_BALANCE	CUS_RATING	CUS_DUE
23	Pankaj	Bihar	2300	230	3	320
Fragment Name : CUS_HAR		**Location: Hisar**		**Node:HRS**		
CUS_ID	CUS_NAME	CUS_SATE	CUS_DEPOSIT	CUS_BALANCE	CUS_RATING	CUS_DUE
10	Puranchand	Haryana	3000	2000	3	1000
21	Ramlal	Haryana	2000	190	3	280
43	Satbir	Haryana	4500	1000	1	900
Fragment Name : CUS_PUN		**Location: Amritsar**		**Node:PNS**		
CUS_ID	CUS_NAME	CUS_SATE	CUS_DEPOSIT	CUS_BALANCE	CUS_RATING	CUS_DUE
11	Rohit	Punjab	4000	3000	2	1500
33	Rahul	Punjab	3300	450	2	400

9.3.2 Vertical Fragmentation

Vertical fragmentation is the same as decomposition. Vertical fragmentation of a relation or a table can be obtained by dividing the table into a number of sub-tables having disjoint columns.

Relation r can be reconstructed from the fragments by taking the natural join operation. Suppose, now that the company is divided into two departments - customer department and collection department. The two departments are concerned with their respective data only. Therefore, the relation CUSTOMER can be vertically fragmented into two fragments as given below:

Fragment name	Location	Node name	Attributes
CUS_DEPT	Customer Office	CUS	CUS_ID, CUS_NAME, CUS_SATATE
COL_DEPT	Collection Office	COL	CUS_ID, CUS_DEPOSIT, CUS_BALANCE, CUS_RATING, CUS_DUE

The resulting two fragment relations are:

Fragment name: CUS_DEPT Location: Customer Office Node:CUS		
CUS_ID	**CUS_NAME**	**CUS_SATE**
10	Puranchand	Haryana
11	Rohit	Punjab
21	Ramlal	Haryana
23	Pankaj	Bihar
33	Rahul	Punjab
43	Satbir	Haryana

Fragment name: COL_DEPT		**Location: Collection Office**		**Node: COL**
CUS_ID	**CUS_DEPOSIT**	**CUS_BALANCE**	**CUS_RATING**	**CUS_DUE**
10	3000	2000	3	1000
11	4000	3000	2	1500
21	2000	190	3	280
23	2300	230	3	320
33	3300	450	2	400
43	4500	1000	1	900

Generally, vertical fragmentation is accomplished by adding a special attribute called a tuple-id to the scheme R (CUS_ID in our case). A tuple-id is a physical or logical address for a tuple. Since each tuple in r must have a unique address, the tuple-id attribute is a key for the augmented scheme.

To reconstruct the original deposit relation from the fragments, we compute

CUSTOMER = (CUS_DEPT ∞ COL_DEPT)

Note that the expression (CUS_DEPT ∞ COL_DEPT) is special form of natural join. The join attribute is CUS_ID. Since the tupled-value represents an address, it is possible to pair a tuple of CUS_DEPT with corresponding tuple of COL_DEPT by using the address given by the CUS_ID value. This address allows direct retrieval of the tuple without the need for an index. Thus, this natural join may be computed much more efficiently that typical natural joins.

Although the tuple-id attribute is important in the implementation of vertical portioning, it is important that this attribute not be visible to users. If users are given access to tuple-ids, it becomes impossible for the system to change tuple addresses. Furthermore, the accessibility of internal addresses violates the notion of data independence, one of the main virtues of the relational model.

9.3.3 Mixed Fragmentation

A relation can also be fragmented both horizontally as well as vertically depending on the application. In such cases both horizontal and vertical fragmentation criteria are specified. Suppose in our example of the CUSTOMER relation, we require each department data separately in the two separate offices at the state headquarters. The required fragments will be:

Fragment name	Location	Horizontal criterion	Node name	Attributes
CUS_BHR_CUS	Patna	CUS_SATE="Bihar"	BHRCUS	CUS_ID, CUS_NAME, CUS_STATE
CUS_BHR_COL	Gaya	CUS_SATE="Bihar"	BHRCOL	CUS_ID, CUS_DEPOSIT, CUS_BALANCE, CUS_DUE
CUS_HAR_CUS	Hisar	CUS_SATE="Haryana"	HARCUS	CUS_ID, CUS_NAME, CUS_STATE
CUS_HAR_COL	Karnal	CUS_SATE="Haryana"	HARCOL	CUS_ID, CUS_DEPOSIT, CUS_BALANCE, CUS_DUE
CUS_PUN_CUS	Amritsar	CUS_SATE="Punjab"	PUNCUS	CUS_ID, CUS_NAME, CUS_STATE
CUS_PUN_COL	Bhatinda	CUS_SATE="Punjab"	PUNCOL	CUS_ID, CUS_DEPOSIT, CUS_BALANCE, CUS_DUE

The resulting fragments are:

Fragment name: CUS_BHR_CUS
Location: Patna
Node:BHRCUS

CUS_ID	CUS_NAME	CUS_SATE
23	Pankaj	Bihar

Fragment name: CUS_BHR_COL		Location: Gaya		Node: BHRCOL
CUS_ID	**CUS_DEPOSIT**	**CUS_BALANCE**	**CUS_RATING**	**CUS_DUE**
23	2300	230	3	320

Fragment name: CUS_HAR_CUS Location: Hisar Node:HARCUS		
CUS_ID	**CUS_NAME**	**CUS_SATE**
10	Puranchand	Haryana
21	Ramlal	Haryana
43	Satbir	Haryana

Fragment name: CUS_HAR_COL		Location: Karnal		Node: HARCOL
CUS_ID	**CUS_DEPOSIT**	**CUS_BALANCE**	**CUS_RATING**	**CUS_DUE**
10	3000	2000	3	1000
21	2000	190	3	280
43	4500	1000	1	900

Fragment name: CUS_PUN_CUS Location: Amitsar Node:PUNCUS		
CUS_ID	**CUS_NAME**	**CUS_SATE**
11	Rohit	Punjab
33	Rahul	Punjab

Fragment name: CUS_PUN_COL		Location: Bhatinda		Node: PUNCOL
CUS_ID	**CUS_DEPOSIT**	**CUS_BALANCE**	**CUS_RATING**	**CUS_DUE**
11	4000	3000	2	1500
33	3300	450	2	400

9.4 DATA REPLICATION

In simple words, Replication is making a copy of the relation. When a relation r is modified or replicated, a copy of relation r is stored in other sites. The copies may be kept at only a few selected sites or each site may keep a copy. In case each site of the system has a copy of the relation it is known as full replication.

Replication is useful in improving the availability of data. The most extreme case is Replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database. This can improve availability remarkably because the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries, because the result of such a query can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module.

The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent. This is especially true if many copies of the database exist.

Full Replication makes the concurrency control and recovery techniques more expensive than they would be if there were no replication. At the other extreme of full replication is to have no replication - that is, each fragment is stored at exactly one site. In this case all fragments must be disjoint, except for the repetition of primary keys among vertical (or mixed) fragments. This is sometimes called unreplicated database.

Between these two extremes, a wide spectrum of partial replication of the data exists—that is, some fragments of the database may be replicated whereas others may not. The number of copies of each fragment can range from one up to the total number of sites in the distributed system. A special case of partial replication is occurring heavily in applications where mobile workers—such as the sales force, financial planners, and claims adjustors—carry partially replicated databases with them on laptops and personal digital assistants and synchronize them periodically with the server database. A description of the replication of fragments is sometimes called a replication schema.

9.4.1 Advantages and Disadvantages of Replication

- ***Increased Parallelism:*** In the case where the majority of access to the relation r results in only the reading of the relation, the several sites can process queries involving r in parallel. The more replicas of r there are, the greater the chance that the needed data is found in the site where the transaction is executing, Hence, data replication minimized movement of data between sites.
- ***Availability:*** If one of the sites containing relation r fails, then the relation r may be found in another site. Thus the system may continue to process queries involving r despite the failure of one site.
- ***Increase overhead on Update:*** The system must ensure that all replicas of a relation r are consistent since otherwise erroneous computations may result. This implies that whenever r is updated, this update must be propagated to all sites containing replicas, resulting in increased overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary that transactions assure that the balance in a particular account agrees in all sites.

9.5 DATA ALLOCATION

Every fragment or each copy of a fragment must be reflected to a particular site in the distributed system. This process is called Data Distribution (or Data Allocation). The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site and if most transactions are retrieval only, a fully replicated database is a good choice. However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication. Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.

There are primarily three data allocation strategies:

- ***Centralized:*** In this strategy the entire database is stored at one site.
- ***Partitioned:*** The database if fragmented into disjoint parts and stored on one or more sites.
- ***Replicated:*** Multiple copies of each fragment are stored at several sites.

9.6 QUERY PROCESSING IN DISTRIBUTED DATABASES

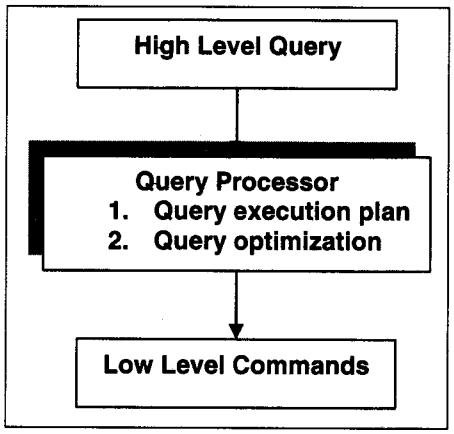
The way queries are processed in a DDBMS is different from the way it is processed in local database systems. The main issue here is the communication costs of processing a distributed query. The query processing system in DDBMS attempts to minimize the amount of network transmission while maximizing the parallel execution of queries on various data sites.

Generally in a distributed system, some additional factors can complicate further query processing. The first is the cost of transferring data over the network. This data includes intermediate files that are transferred to other sites for further processing, as well as the final result files that may have to be transferred to the site where the query result is needed. Although these costs may not be very high if the sites are connected via a high-performance local area network, they become quite significant in other types of networks. Hence, DDBMS query optimization algorithms consider the goal of reducing the amount of data transfer as an optimization criterion in choosing a distributed query execution strategy.

9.6.1 Semijoin

One way of carrying out query processing in a distributed database system is to use semijoin. The main logic behind distributed query processing using the semijoin operations is to reduce the number of tuples in a relation before transferring it to another site. Infact, the idea is to send the joining column of one relation R to the site where the other relation S is located; this column is then joined with S. Following that, the join attributes, along with the attributes required in the result, are projected out and shipped back to the original site and joined with R. Hence, only the joining column of R is transferred in one direction, and a subset of S with no extraneous tuples or attributes is transferred in the other direction. If only a small fraction of the tuples in S participate in the join, this can be quite an efficient solution to minimizing data transfer.

Consider a high level query submitted to the query processor. The query is de-fragmented into simpler low level query commands as shown below.



The high level query can be executed in a variety of different ways called query execution plans. Some of these plans achieve greater cost benefit as far as network movement is concerned.

Assume that following is the high level query submitted to the query processor.

SELECT SNAME **FROM** STUDENT, COURSE

WHERE STUDENT.ROLLNO = COURSE.ROLLNO **AND** CREDIT > 3

Two different strategies can be devised for the execution of this query as listed below.

Plan-1: SELECT SNAME FROM

(SELECT * FROM STUDENT JOIN COURSE

WHERE student.rollno=course.rollno and credit > 3)

Plan-2: SELECT SNAME FROM

(SELECT * FROM STUDENT WHERE

rollno = (SELECT rollno WHERE credit >3))

To select one from the above execution plans we see that since the second plan does not involve Cartesian product of the relations, it should be preferred.

Check Your Progress

Fill in the blanks:

1. A distributed database management system (DDBMS) is a software system that manages a distributed database while making the distribution ..... to the user.
2. In a Distributed processing system the data comes from a ..... database systems but the processing is performed on more than one sites.
3. Vertical fragmentation of a relation or a table can be obtained by dividing the table into a number of sub-tables having ..... columns.
4. The disadvantage of full replication is that it can slow down .....

9.7 LET US SUM UP

A distributed database is a database which is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Distributed system can be taught of as a partnership among independent but cooperating. Location transparency means that users should not need to know at which site any given piece of data is stored, but should be able to behave as if the entire database were stored at their own site. A system supports data fragmentation if data or file can be divided into pieces (fragments) for physical storage purpose. Replication transparency means that basic idea is that a given logical object, say a given account record, may be represented at physical level by many distinct copies. ANSI/APARC Architecture It is a 3-level architecture based on data organization When processing a query, distributed DBMS and parallel DBMS analyze the potential parallelism of the request and make query plans using Extended Dataflow Graph (EDG) along with the Engineering Model that the database use.

9.8 KEYWORDS

Distributed Database: A database which is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

Location Transparency: A mechanism through which users do not need to know at which site any given piece of data is stored, but should be able to behave as if the entire database were stored at their own site.

Data Fragmentation: A system supports data fragmentation if data or file can be divided into pieces (fragments) for physical storage purpose.

Data Replication Transparency: It means that basic idea is that a given logical object, say a given account record, may be represented at physical level by many distinct copies.

ANSI/APARC Architecture: It is a 3-level architecture based on data organization.

EDG: An EDG is a self-scheduling structure such that once the START signal is sent, the execution of the EDG will run to completion by itself without any external control. The control of the execution of the EDG is completely data driven.

9.9 QUESTIONS FOR DISCUSSION

1. What are the two forms of parallelism that can be applied to DBMSs?
2. Explain parallel query processing. How does the parallel query feature help to improve performance?
3. Explain query optimization and cost based query optimizers.
4. Explain Parallel query optimization.

5. Explain the need for distributed DBMS. Enumerate the advantages offered by a distributed DBMS.
6. Explain the following terms in relation with distributed database
   (a) Transparency
   (b) Data fragmentation
   (c) Data replication
7. Discuss what are the major issues concerning distributed DBMS.
8. Draw the diagram of a basic architecture of a distributed DBMS and explain its components.
9. Explain query processing in distributed database.

Check Your Progress: Model Answers

1. Transparent
2. Centralized
3. Disjoint
4. update operations

9.10 SUGGESTED READINGS

Elisa Bertino, *Distributed and Parallel Database Object Management*, 2001, Springer-Verlag New York

Ceri & Pelagatti, *Distributed Databases - Principles and Systems*, 1985, McGraw-Hill

Clement Yu, Weiyi Meng, *Principles of Database Query Processing for Advanced Applications*

M. Tamer Özsu, Patrick Valduriez, *Principles of Distributed Database Systems*, Second Edition, Prentice Hall, 1999

David Bell, Jane Grimson, *Distributed Database Systems*, Addison Wesley, 1992