M.Sc. (IT) Previous Year

MIT-07

ORACLE

मध्यप्रदेश भोज (मुक्त) विश्वविद्यालय – भोपाल

MADHYA PRADESH BHOJ (OPEN) UNIVERSITY - BHOPAL

Reviewer Committee

1. Dr. Sharad Gangele
   Professor
   R.K.D.F. University, Bhopal (M.P.)
2. Dr. Romsha Sharma
   Professor
   Sri Sathya Sai College for Women,
   Bhopal (M.P.)
3. Dr. K. Mani Kandan Nair
   Department of Computer Science
   Makhanlal Chaturvedi National University of Journalism and Communication, Bhopal (M.P.)

Advisory Committee

1. Dr. Jayant Sonwalkar
   Hon'ble Vice Chancellor
   Madhya Pradesh Bhoj (Open) University,
   Bhopal (M.P.)
2. Dr. L.S. Solanki
   Registrar
   Madhya Pradesh Bhoj (Open) University,
   Bhopal (M.P.)
3. Dr. Kishor John
   Director
   Madhya Pradesh Bhoj (Open) University,
   Bhopal (M.P.)
4. Dr. Sharad Gangele
   Professor
   R.K.D.F. University, Bhopal (M.P.)
5. Dr. Romsha Sharma
   Professor
   Sri Sathya Sai College for Women,
   Bhopal (M.P.)
6. Dr. K. Mani Kandan Nair
   Department of Computer Science
   Makhanlal Chaturvedi National University of Journalism and Communication, Bhopal (M.P.)

COURSE WRITERS

Kavita Saini, Associate Professor, Galgotias University, Greater Noida
Units (1.0-1.6, 1.12-1.16, 2.0-2.1, 2.3, 2.5-2.9, 3.0-3.2.2, 3.2.3, 3.3-3.15, 4.0-4.2, 4.6-4.10, 5.0-5.1, 5.6, 5.8-5.12)
Umang Garg, Assistant Professor, Dept. of Computer Science and Engineering, Graphic Era Hill University, Dehradun
Units (1.7-1.11, 2.2, 2.4, 4.3, 4.4, 4.5, 5.2-5.5, 5.7)
Rohit Khurana, Faculty and Head, I.T.L. Education Solutions Ltd., New Delhi
Unit (4.3.1)

Published by Registrar, MP Bhoj (open) University, Bhopal in 2020

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

SYLLABI-BOOK MAPPING TABLE

Oracle

Syllabi	Mapping in Book
UNIT - I: **Getting Started With Oracle:** Overview of RDBMS, Getting Started, Module of Oracle, Invoking SQLPLUS, Data Types, Data Constraints. **Operators in SQL**: Precedence of Operators, Types of Operators8, SQL * Plus Functions, Types of Functions. **Database Objects:** Introduction to Synonyms, Introduction to Sequences, Alternating of Sequence, Introduction to Indexes. **Data Integrity**: Purpose, Prerequisites, Keywords and Parameters, references clause, SCOPE REF Constraints, VALIDATE NOVALIDATE, Using Indexes to Enforce Constraints.	**Unit-1:** Oracle, Database Objects and Data Integrity **(Pages 3-59)**
UNIT - II **Formatting SQL * Plus Reports and Commands:** Formatting Columns, Clarifying Your Report with Spacing and Summary Lines, Defining Page and Report Titles and Dimensions, Storing and Printing Query Results, SQL *PLUS Commands. **SQL * Loader:** Introduction to SQL * Loader, Bad Files, Discard File, The Control File. **Accessing Remote Database**: Introduction to Database Links, Using Database Links for Remote Queries, Dynamic Links: Using SQL PLUS Copy Command, Connecting to Remote Database.	**Unit-2:** SQL* Plus Reports, Commands, Loader and Accessing Remote Database **(Pages 61-111)**
Unit - III **Overview of PL/SQL**: Understanding the Main Features of PL/SQL, The ORACLE Database Server, Advantages of PL/SQL. **Procedures, Functions and Packages:** Stored Procedures, How to Create and Execute Procedures? Where to Store Procedures? Stored Functions, How to Create & Execute Functions, Where to store Functions, Where do Procedures and Functions Reside?	**Unit-3:** Overview of PL/SQL **(Pages 113-150)**
UNIT - IV **Triggers:** Introduction to Database Triggers, Required System Privileges, Parts of A Trigger, Types of Trigger, Instead of Trigger, Enabling and disabling Trigger. **Object Relational Databases**: Enhancements, Features of Object-Oriented Programming, Introduction to Object Views, Manipulating Data through Object View, Introduction to Methods. **Collections (Nested Tables & Varying Arrays):** Introduction to Varying Arrays, Creating of Varying Arrays, Maintaining of Varying Arrays, Introduction to Nested Tables.	**Unit-4:** Triggers, Object Relational Database, Nested Tables and Varying Arrays **(Pages 151-202)**
UNIT - V **Using Large Objects:** Available Data Types, Specifying Storage for LOB Data, Manipulating and Selecting LOB Values. **Introduction to Web Enabled Database**: Role of SQL, Understand the Role of Java and WebDB, Introduction to web architecture. **A Brief Introduction About Database Administration**: Creating a Database, Creating and Managing Rollback Segments, When Rollback Information is Required, Rollback Segment States, What is Backup and Recovery, How recovery works.	**Unit-5:** Introduction to Web enabled database and database administration **(Pages 203-245)**

CONTENTS

UNIT 5 INTRODUCTION TO WEB ENABLED DATABASE AND DATABASE ADMINISTRATION 203–245

INTRODUCTION

NOTES

The Oracle is an Object Relational Database Management System (ORDBMS) produced and marketed by Oracle Corporation. Users of the Oracle Databases refer to the server side memory structure as the SGA or System Global Area. The SGA typically holds cache information, such as data buffers, SQL commands and user information. In addition to storage, the database consists of online redo logs or logs which hold transactional history. Processes can in turn archive the online redo logs into archive logs (offline redo logs), which provide the basis for data recovery and for some forms of data replication. The Oracle DBMS can store and execute stored procedures and functions within it. PL/SQL or Procedural Language/Structured Query Language is Oracle Corporation's procedural extension language for SQL and the Oracle relational database. PL/SQL can invoke such code objects and/or provide the programming structures for writing them. PL/SQL supports variables, conditions, loops and exceptions. PL/SQL program units (essentially code containers) can be compiled into the Oracle Database so that programmers can embed PL/SQL units of functionality into the database directly. They also can write scripts containing PL/SQL program units that can be read into the database using the Oracle SQL*Plus tool. The Oracle RDBMS stores data logically in the form of tablespaces and physically in the form of data files. The Program Global Area or PGA memory area of an Oracle Instance contains data and control information for Oracle's Server processes. The size and content of the PGA depends on the Oracle Server options installed.

This book, *Oracle* is divided into five units that follow the self-instruction mode with each unit beginning with an Introduction to the unit, followed by an outline of the Objectives. The detailed content is then presented in a simple but structured manner interspersed with Check Your Progress Questions to test the student's understanding of the topic. A Summary along with a list of Key Terms and a set of Self-Assessment Questions and Exercises is also provided at the end of each unit for recapitulation.

UNIT 1 ORACLE, DATABASE OBJECTS AND DATA INTEGRITY

NOTES

Structure

1.0 INTRODUCTION

Oracle is a program or set of processes running in a computer's operating system. These processes manage the storage and access of data. SQLplus is Oracle's tool used to access the database and create programs. SQLplus has a command line interface, which helps to access the database and write stored procedures. It also helps to run SQL commands to retrieve data and run scripts of built-in SQLplus commands. Simply referred to Oracle (or Oracle RDBMS) is a relational database management system (RDBMS).

In databases, a synonym is an alias or alternate name for a table, view, sequence, or other schema object. They are used mainly to make it easy for users to access database objects owned by other users. They hide the underlying object's

NOTES

identity and make it harder for a malicious program or user to target the underlying object. Because a synonym is just an alternate name for an object, it requires no storage other than its definition. When an application uses a synonym, the DBMS forwards the request to the synonym's underlying base object.

Data integrity is the maintenance of, and the assurance of, data accuracy and consistency over its entire life-cycle and is a critical aspect to the design, implementation, and usage of any system that stores, processes, or retrieves data. Oracle uses integrity constraints to prevent invalid data entry into the base tables of the database. You can define integrity constraints to enforce the business rules that are associated with the information in a database.

In this unit you will study about the Overview of relational database management systems, introduction to Oracle, data constraints, operators on Oracle, introduction to synonyms and sequence, introduction to indexes and data integrity.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the concept of relational database management system
- Describe the modules of Oracle
- Mention various data constraints
- Describe the operators in Oracle
- Discuss the synonyms and sequences
- Explain the concept of indexes and data integrity

1.2 OVERVIEW OF RELATIONAL DATABASE MANAGEMENT SYSTEMS

RDBMS is the acronym of Relational Database Management System. The idea of RDBMS came from the Database Management System. To understand the relational database management system, it is necessary to know about the database management system.

Major Models for Data Management

The major model proposed for management of data are as follows:

- Hierarchical Model
- Network Model
- Relational Model

Hierarchical Model

This model was proposed before network and relational models. Data in hierarchical model is represented as a tree structure. Representation in this model is depicted in Figure 1.1.

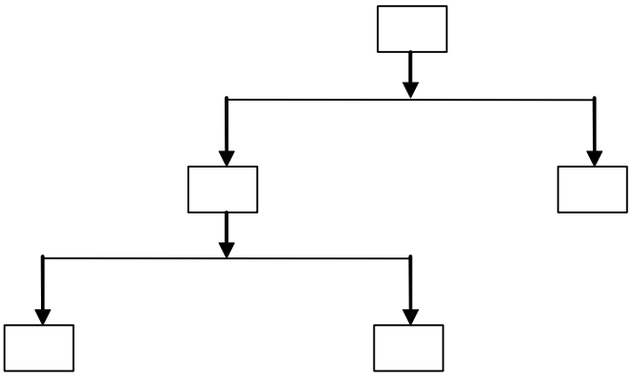


Fig. 1.1 *Hierarchical Model*

The biggest disadvantage of this model is that it is not suitable for a complex data structure. Other disadvantage is that it supports only one to one or one to many relationships. Many to many relationships could not be defined. Another disadvantage of the hierarchal model is that the relationship should be defined before creating the tree structure. So it is very inflexible data model.

Network Model

This model has improvement over hierarchical model and supports one to one, one to many and many to many relationships between data. The network model is depicted in Figure 1.2.

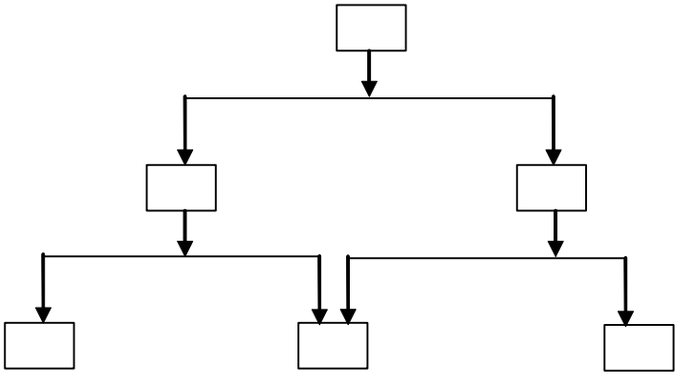


Fig. 1.2 *Network Model*

The disadvantage of network model is that the relationship should be defined before creating the data structure. As relationship increases the data structure becomes complex.

Relational Model

A relational model is introduced by Dr. E.F. Codd. Data in this model is represented in two dimensional structures called a table or a relation. A relational model is represented in Figure 1.3.

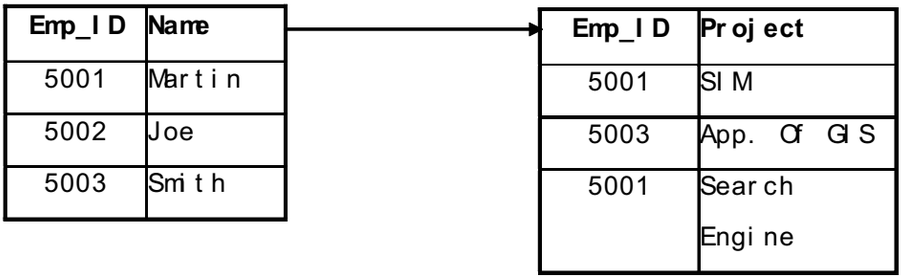


Fig. 1.3 *Relational Model*

As stated earlier, in hierarchal and network models, relationships are required to be pre-defined. It becomes very complex to add new relationships, updating or deleting existing relationships in the existing data structure. On the other side, relationships in relational model can be added, modified or updated on the existing data with effecting other data or relationships. Due to the flexibility of the relational model, it is a useful and popular model.

Introduction to RDBMS

A database is a collection of interrelated data stored in two-dimensional structures. It helps organizations to keep records of inventory, employees details, account payable and receivable, and university data, etc.

Once data is stored in a database, it should be managed in future. A database management includes:

- Creating a new database
- Adding, deleting or updating database
- Retrieval of stored data
- Putting some constraints on data (e.g., constraint on ID for uniqueness) to maintain data integrity

A Relational Database Management System (RDBMS) is a collection of database and stored procedures. It enables you to store, extract and manage important information from a database. It is a software that is used to maintain data security and data integrity in a structured database.

DBMS Components

The components of a DBMS are listed as follows:

- Hardware
- Software
- People
- Network

RDBMS helps in maintaining and retrieving data in different forms. There are various tools available for RDBMS such as those listed as follows:

- Oracle
- INGRES
- Sybase

- Microsoft SQL Server
- MS-Access
- IBM-DB-II
- MySQL

NOTES

Twelve Rules of E. F. Codd

Edgar F. Codd has defined twelve rules to define the requirements and features for a relational model. These rules are as follows:

1. The information rule
2. The guaranteed access rule
3. Systematic treatment of null values
4. Active online catalog based on the relational model
5. Data sublanguage rule
6. The view updating rule
7. High-level insert, update, and delete
8. Physical data independence
9. Logical data independence
10. Integrity independence
11. Distribution independence
12. Non-subversion rule

The description of these rules is given as follows:

1. **The information rule:**

   This rule simply requires all information to be represented as data value in a rows and columns of tables.

2. **The guaranteed access rule:**

   Every data value in a relational database should be guaranteed accessible by specifying a combination of the table name, primary key value and column name.

3. **Systematic treatment of null values:**

   The DBMS must support systematic treatment of null values. A missing, unknown or inapplicable data should be represented as a null value. They must be distinct from zero or space and must be independent of data type.

4. **Active online catalog based on the relational model:**

   The database management system must support the catalog-based model. For example, a DBMS supports a system catalog. A system catalog is a collection of tables that is maintained by the DBMS itself. It holds the description of table structure. It basically stores data about data and users must be able to access the catalog using the same query language that is used to access the database's data.

5. **Data sublanguage rule:**

   This rule specifies that the system must support a language which could perform the following functions :

NOTES

- Data definition
- View definition
- Data manipulation
- Data security and integrity constraints
- Transaction management operations

6. **The view updating rule:**

   All views that are theoretically updatable must be updated through the system (a view is a virtual table).

7. **High-level insert, update and delete:**

   This rule states that the DBMS must support insert, update and delete operations on a set. The Select operation could retrieve a set of rows, and in the same manner Modify and Delete operations should also be done of a set of rows.

8. **Physical data independence:**

   This rule states that application programs must be unaffected when physical access methods or storage structures are altered.

9. **Logical data independence:**

   This rule states that any change to the logical level of table must not affect the application programs in accessing the data.

10. **Integrity independence:**

    To maintain data integrity, a database language must support integrity constraints. Integrity constraints must be storable in catalog and cannot be bypassed.

11. **Distribution independence:**

    The database must allow manipulation of distributed data located on other computer system. An application must not be affected when data is first distributed or when it is redistributed.

12. **Non-subversion rule:**

    This rule states that different levels of language can not subvert or bypass the integrity rules and constraints.

Basic Terms used for database

The terms used while discussing databases are as follows:

- Table
- Field and Domain
- Records or tuple
- **Table:** A table is represented in two-dimensional structure containing rows and columns. It contains interrelated data, for example, an employee table contains data about employee only i.e. Emp_ID, name, designation, etc. a table is also termed as a relation. Such a table is depicted in Table 1.1:

NOTES

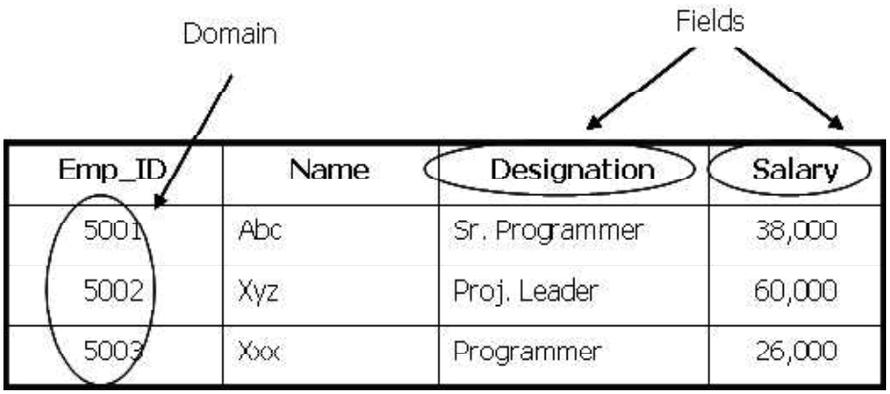
Table 1.1 *Sample Table*

Emp_ID	Name	Designation	Salary
5001	Tom	Sr. Programmer	38,000
5002	Merlisa	Proj. Leader	60,000
5003	George	Programmer	26,000

- **Field and Domain:** A field specifies what data will be stored in a particular column, such as, name is a field in an employee table in which employee names are stored.

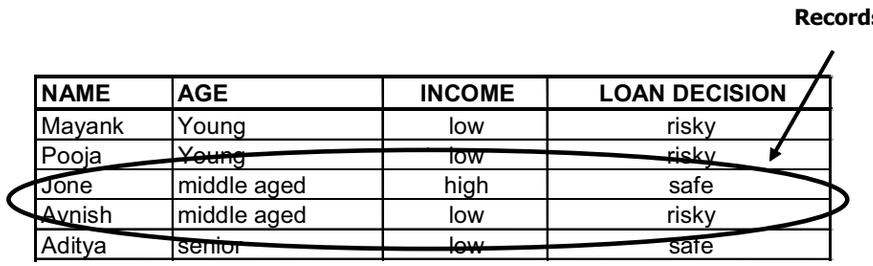
  A field is a column heading that specifies what type of data you could store, but the data stored in a particular column is a pool of the same type of data such as all employee names stored in the name field. These employee names are a domain of name. A domain contains same type of information. Domains and fields are depicted in Table 1.2.

Table 1.2 *Sample Field or Domain*



- **Record or tuple:** Related data stored in a row is termed as a record or a tuple. It contains different types of information. Table 1.3 contains records and tuples.

Table 1.3 *Sample Record or Tuple*



Benefits of Database Management System

Database management system has many features over the file system and other data models. Few benefits are described as follows:

- **Data Independence:** In the past data and program were bounded together. Any change in data lead to change the entire application. One of the most significant benefits of database is its data independency. Data and application now could be two different entities which provide the flexibility to change data without changing the application.
- **Controlled Data Redundancy:** The amount of data redundancy could be reduced by using database. A centralized database could be shared by all the departments in the organization which helps in reducing data redundancy.
- **Efficient Data Access:** A database provides multiple views of database in efficient manner. The same database could be seen by different users with different criteria or a search condition. A DBMS provides sophisticated techniques to retrieve the stored data.
- **Data Integrity:** Data integrity refers to the data accuracy. For example, all the students in a university must have a unique enrollment number, salary of employee should not be negative and so on. Database provides various constraints as primary key, foreign key and unique key etc. which helps in maintaining data integrity.
- **Data Security:** An organization may have many users of database so data security is a major issue. All the data should not be accessible to all the users. Data access permission could be assigned to different users as per their role and responsibilities in the organization to maintain data security.
- **Data Sharing:** The center database could be stored on a server (database server), which could be accessed concurrent by the different users at the same time.
- **Data Consistency:** Any changes made by one user reflect to the other users which help in maintaining data consistency.
- **Backup and Recovery:** Database provides backup feature to take backup of data store in a primary disk. If case primary disk fails the data could be recovered.

Application of DBMS in Various Fields

In day-to-day life, various applications are in use. Few applications are written where database is used:

- **Banking:** For account holder information, amount withdraw and deposit and other transactions
- **Airlines:** For reservations , cancellation , fare detail and airline schedules
- **Universities:** For student registration, examination, fee detail, course detail and other information
- **Manufacturing:** For inventory, production, sale and purchase orders

- **Human Resources:** Employee records, salaries, tax deductions, allowances
- **Multimedia application**
- **Real time application**
- **Graphical Information System (GIS)**

NOTES

1.3 INTRODUCTION TO ORACLE

Oracle is a secure, portable and powerful database management system of Oracle Corporation. It is also termed as Oracle Database. It is compatible and connectable with almost all operating systems and machines.

Oracle database is based on relational data model and a non-procedural language called structure query language (SQL). This is a tool that supports storage, management and organization of the data.

1.3.1 Modules of Oracle

Oracle has various products as listed here:

- SQL *Plus and Functions
- Oracle Form Builder
- Oracle Report Builder
- Oracle Graphics

SQL *Plus and Functions: SQLplus is a command line tool that allows a user to execute SQL statements against an Oracle database. Basically it supports two types of SQL:

1. Interactive SQL
2. PL/SQL

Interactive SQL is used to execute various types of SQL commands such as Create, Insert, Update, and Delete to maintain data. SQL command are written on SQL> prompt that gets executed immediately, so it is known as an interactive SQL.

On the other hand PL/SQL can be used to write programs to maintain database. PL/SQL programs are stored permanently to maintain databases.

Oracle Form Builder: Oracle Forms is an Oracle tool that helps to create graphical user interfaces (screens) based on Oracle database. Various input and output forms could be designed to make the application more user friendly. Forms could be designed and customized to add various functionalities by using controls such as text box, combo boxes, radio buttons, and list of values. PL/ SQL procedures, triggers and functions could be written in form's code editor.

Oracle Report Builder: Oracle Reports is an Oracle tool that helps to create and organization reports. A report builder includes a query builder, default report templates, default layouts and integrated chart builder.

Oracle Graphics: An Oracle graphics tool in used to develop graphs and charts. This way graph and charts could be displayed in Oracle forms and reports to represent any data.

NOTES

1.3.2 Structured Query Language (SQL)

SQL is a query language used for all database relation management systems. It is a standard language for all RDBMSs. SQL can be classified every sub-various plays its own sole and cater different purpose SQL as follows:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

Data Definition Language (DDL): Data Definition Language commands are used for creating, modifying and removing database objects, the object could be a table cursor view trigger used or a sequence.

Data Definition Language commands are as follows:

- `CREATE`
- `ALTER`
- `DROP`

Various objects can be created using DDL commands as :

```
CREATE  TABLE
CREATE  VIEW
CREATE  FUNCTION
CREATE  PROCEDURE
ALTER  VIEW
ALTER  TABLE
DROP  TABLE
DROP  VIEW
```

Data Manipulation Language (DML): Once a table or other object is created using Data Definition Language, Data Manipulation Language (DML) commands are used to insert, manipulate and access data. DML commands help in inserting, updating, deleting and searching of data. Data manipulation language statements are as follows:

- `INSERT`
- `DELETE`
- `SELECT`
- `UPDATE`

Data Control Language (DCL): Data Control Language commands allow authorized database users to share data with other users. Shared data could be accessed or manipulated by other users as per the permission granted to those users.

The data manipulation language statements are as follows:

- GRANT
- REVOKE

NOTES

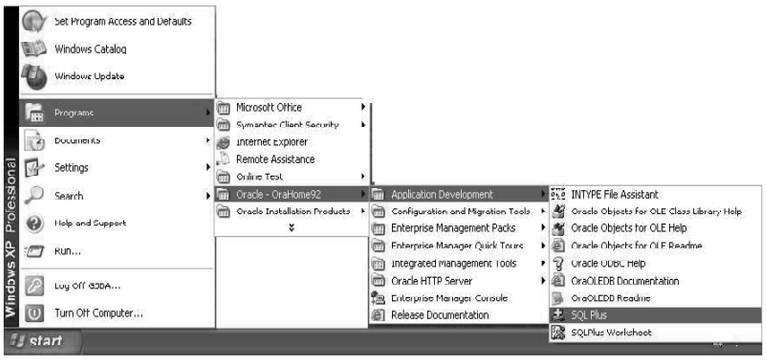
Transaction Control Language (TCL): Various transactions are being done by different users. These transactions then could be saved permanently or can be called by the user. TCL commands manage changes made by DML statements. Transaction Control Statements are as follows:

- COMMIT
- ROLLBACK
- SAVEPOINT

1.3.3 Getting Started with SQL*Plus

To work with SQL*Plus, Oracle should to be installed on computer systems. The following steps are required to follow to invoke SQL*Plus:

1. Click on `Start` button
2. Point on `Programs`
3. Point on `Oracle` → `OraHome92` → `Application Development`
4. Click on `SQL Plus`



Or

Double click SQL*Plus shortcut on the desktop.

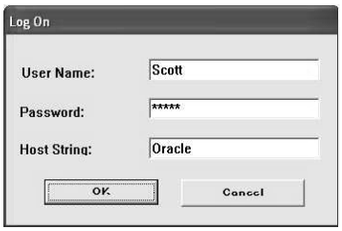
Oracle is very secure and only authorized users are able to access the database. These users in Oracle are created and managed by the database administrator (DBA). A DBA creates or drops users and grants or revokes the privileges to them.

Oracle provides few default users:

- SYS
- System
- Scott

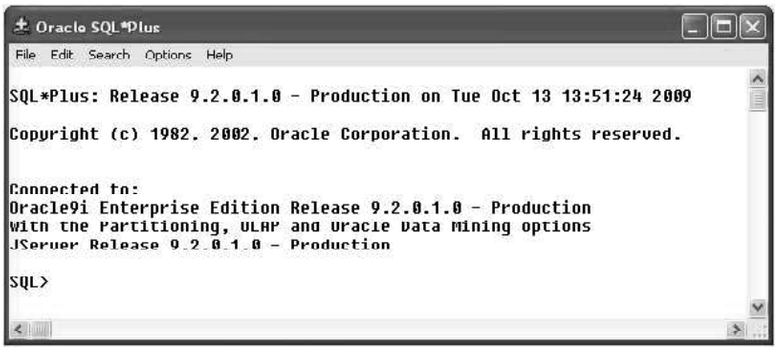
The password of `scott` user is `tiger` that you can use to connect to the Oracle database. After following steps to start SQL *Plus the **Log On** screen will appear shown as follows:

NOTES



5. Use the User Name Scott, password Tiger and host string Oracle (Consult your lab instructor for host string name).
6. Click on 'OK' button.

After clicking OK button, the following screen will be displayed that contains the information about the SQL *Plus product.



SQL > is the SQL prompt where all the interactive SQL commands are entered and executed.

Basic Guidelines for SQL Commands

The basic guidelines for SQL commands are as follows:

- Every SQL statement must be terminated with the semicolon (;)
- SQL commands are not case sensitive.
- SQL command could be written either in a single line or in multiple lines.
- Values stored in table are case sensitive (i.e. – 'computer' is not equal to computer').

1.3.4 Data Types in Oracle

When you define any table, it is required to specify the data type of fields. The main categories of data types are as follows:

- Number
- Character
- Date/Time
- Binary type
- Image or pictures

Data types and their size are explained and discussed in Table 1.4.

NOTES

Table 1.4 *Data Types*

Data Type	Size	Explanation
Char (size)	Maximum size of 2000 bytes	It is a Fixed length character data type. The default length of char is 1. Any space left after entered data is filled with blanks. For example Code Char (4) Such as 'A001','A3 ', …
Varchar2 (size)	Maximum size of 4000 bytes	It is a variable-length string data type. Varchar2 saves space if data stored is less than the specified size. For example Code Varchar2 (4) Such as 'A001','A3', …
Long	Maximum size of 2 GB	It is used to store large amounts of variable-length text. Only one column can be defined as LONG in a table. A LONG value cannot be used in where, order by and group by clause.
Raw (size)	Maximum size of 2000 bytes	It is used to store variable length binary data in a column.
long raw (size)	Maximum size of 2 GB	It is also used to store variable length binary data in a column up to 2 gigabytes.
Number (p, s)	Precision can range from 1 to 38. Scale can range from -84 to 127.	The NUMBER data type is used to store numeric value. This value could be negative number, positive number, a fixed number, or a floating point numbers. Where p is the precision and s is the scale. For example Salary number (7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal. Such as 43253.50, 434.7, …

Data Type	Size	Explanation
Date	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.	This data type is used to store date in a column. Seven bytes fixed for DATE data type. The default format is 'DD-MON-YY'. For each DATE data type, the following information is stored: • Month • Day • Year • Hour • Minute • Second For example date_of_join Date '23-Jan-09'

NOTES

Oracle also provides few more data types such as the follows:

- CLOB (**C**haracter **L**arge **O**bject)
- NCLOB (**N**ational **C**haracter **L**arge **O**bject)
- BLOB (**B**inary **L**arge **O**bject)
- BFILE (Pointer to binary file on disk)

1.4 DATA CONSTRAINTS

It is very important that whatever you store into your tables is as per the need of your organization. No false or incorrect data is stored by the user even intentionally or accidentally. Constraints are the restriction that you could put on your data to maintain data integrity. For example, employer's salary should not be negative value, two students should not have the same enrollment number etc.

The constraints help in maintaining data integrity which is one of the rules defined by E.F. Codd. Constraints could be specified when a table is created or even after the table is created with the `ALTER TABLE` command.
Oracle provides various types of constraints as listed here:

- Primary key
- Foreign key or reference key
- Not null
- Unique
- Check
- Default

Constraint could be defined at column level or at the table level. The only difference between these two is the syntax of these two.

Not Null Constraint:

In database, `NULL` is a special value that is different from zero, space or blank. It represents an unknown value for the column.

The `NOT NULL` constraint ensures that the value in column is not missing (`NULL`). This constraint enforce user to enter data into a specified column. A column with this constraint could have duplicate values but could not be null or empty.

You must have created your e-mail ID. When you create an e-mail ID, it is mandatory to fill certain entries (the field with *); those fields are the fields with the not null constraint.

The following example creates a table book with the `NOT NULL` constraint with the structure as shown:

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	
Title	varchar2	40	NOT NULL
Author	varchar2	15	NOT NULL
Price	Number	7,2	

The SQL command to create table with `NOT NULL` constraint is as follows:

```
Create Table Book
(
B_code   varchar (15) ,
Title varchar (40) NOT NULL ,
Author   varchar2 (15) NOT NULL ,
Price number (7,2)
) ;
```

The above SQL command will create a table book where Title and Author have `NOT NULL` constraints. These constraints would make it sure that both the columns have some values during inserting and updating of data to these columns.

`NOT NULL` constraints could be set at column level only.

Unique Constraint

Sometimes it is required that column must have unique values only. The unique constraint ensures that data to the specified column data is not duplicate but it could contain the NULL values. Let us take an example of contact number and e-mail ID; it is not necessary that every student has a contact number and an e-mail ID, if they have that will be unique only.

The following example creates a table student with the `UNIQUE` constraint with the structure as shown:

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	
Name	Varchar	10	
Address	Varchar	35	
E_Mail	Varchar	20	Unique
Mobile	Number	10	Unique

The SQL command to create table with `UNIQUE` constraint is as follows:

```
Create Table Student
    (
       Roll_No  varchar2 (10) ,
       Name     varchar2 (10) ,
       Address  varchar2 (35) ,
       E_Mail          varchar2 (30),
       Mobile         Number (10),
       Unique (E_Mail),     Unique (Mobile)
     ) ;
```

In this example unique constraints are set at table level.

Primary Key Constraint

A primary key constraint is used to uniquely identify each and every record in a table. A primary key has properties of unique and not null constraints.

NOTES

A primary key constraint has the following properties:

- A primary key column allows unique values only.
- It does not allow NULL value in column.
- A primary key column could be used for a reference in another table (child table).

Example 1.1:

The following example creates a table course with the PRIMARY KEY constraint with the structure as shown:

Column Name	Data Type	Size	Constraint
c_code	varchar2	15	Primary Key
c_name	varchar2	15	
duration	number	8	
fee	number	10, 2	

The SQL command to create table with PRIMARY KEY constraint is as follows:

```
Create Table Course
(
C_code      varchar (15)  Primary key,
C_name      varchar (15),
Duration    number (8),
Fee         number (10, 2)
) ;
```

The above command will create table course which contains a primary key field course code. Here primary key constraint will enforce the end user to enter unique and not null values only.

Example 1.2:

The following example creates a table book with the PRIMARY KEY constraint with the structure as follows:

Column Name	Data Type	Size	Constraint
c_code	varchar2	15	Primary Key
c_name	varchar2	15	
duration	number	8	
fee	number	10, 2	

The SQL command to create table with PRIMARY KEY constraint is as follows:

```
Create Table Book
(
B_code   varchar (15) Primary Key,
Title    varchar (40),
Author   varchar2 (15),
```

```
Price     number (7,2)
) ;
```

The above command will create table Book which contains a primary key field book code. This constraint is required to have unique and not null book code in a library.

NOTES

* A table can have only and only one primary key.

Foreign Key Constraint or Reference Key Constraint

A foreign key column in a table derived values from a primary key of another table that helps in establishing relationship between tables.

A table having primary key column is called a Master Table or a parent table and a table with the reference key is known as a Transaction Table or a child table.

A course and book tables created in the primary key constraint section have the primary key columns `C_code` and `B_code` respectively. These columns could be used to as a reference key in another table.

Important points to be remember

- Reference key column in a table must have the same data type be as specified in primary key column in another table.
- Size of data type must be the same or more as defined in a primary key column.
- Name of reference key column could be same or different as defined in primary key column.
- A table may contain more than one reference keys.
- Reference keys column values could be duplicate or not null.
- Reference keys column can have the same values as stored in primary key column.

Suppose that students in any university could be enrolled in the course which are offered by that university. Course table contains the detail of all the courses offered by the university, so `C_code` column in student table must have reference of `C_code` column of course table.

Example 1.3:

The following example creates a table student with the `REFERENCE KEY` constraint with the structure as shown

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	
Name	Varchar	10	
Address	Varchar	35	
C_code	Varchar	15	Reference Key

The SQL command to create table with `REFERENCE KEY` constraint is as follows:

```
Create Table Student
   ( Roll_No      varchar (10) ,
     Name         varchar (10) ,
     Address      varchar (35) ,
     C_code varchar (15) references course (C_code)
 );
```

The above command will create table student which contains a reference key column course code. This column will table reference of course code of course table when record in student table will be inserted or updated by the user.

* A table can have more than one reference keys.

Check Constraint

A check constraint enforces users to enter data as specified condition. For example, marks in any subject should be between the ranges 0 to 100, fee should not be negative, book code must start with 'B' and book price should be between ranges 1 and 1,5000 and employee HRA could not be more than 40 per cent of basic salary and so on.

Example 1.4:

The following example creates a table book with the `CHECK` constraint with the structure as shown

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	Check
Title	varchar2	40	
Author	varchar2	15	
Price	number	7,2	Check

The SQL command to create table with `CHECK` constraint is as follows:

```
  Create Table Book
(
B_code  varchar (15) check ( B_code like 'B%') ,
Title   varchar2 (40) ,
Author   varchar2 (15) ,
Price   number (7,2) check ( Price > 1 and price < =
15000)
) ;
```

The above command will create table Book which contains a check constraints with the field book code and price.

Default Constraint

Some-times the value of any column for every new record is same. To maintain the status of book in a library, either available for issue or not, you must keep the

status of book as 'T' (available) or 'F' (Issued). Every new book purchased for library the status of book is required to be 'T'. Default value concept is suitable for many these types of situations.

The following example creates a table book with the DEFAULT constraint with the structure as shown:

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	
Title	varchar2	40	
Author	varchar2	15	
Price	Number	7,2	
Status	Char	1	Default

The SQL command to create table with DEFAULT constraint is as follows :

```
Create Table Book
(
B_code   varchar (15),
Title varchar (40),
Author  varchar2 (15),
Price number (7,2),
Status   char (1) default 'T'
) ;
```

The above command will create table Book which contains a default constraints with the field status.

Example 1.5:

The following example creates a table student with the multiple constraints with the structure as shown:

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	Primary Key
Name	Varchar	10	Not Null
Address	Varchar	35	
C_code	Varchar	15	Reference Key
Mobile	Number	10	Unique

The SQL command to create this table as follows:

```
Create Table Student
(
      Roll_No     varchar (10) primary key ,
     Name         varchar (10) not null ,
     Address      varchar (35) ,
     C_code varchar (15) references course (C_code)
,
     Mobile number(10) unique
 );
```

NOTES

Check Your Progress

1. List the major models of data management.
2. List three benefits of database management systems.
3. What is SQL?
4. List the forms in which SQL can be classified?
5. What is a constraint?
6. List the various types of constraints as provided by Oracle.

1.5 OPERATORS IN ORACLE: TYPES AND PRECEDENCE

Operators are the special characters that manipulate data items to produce some result. These data items are called operands. Operators are classified into two categories:

1. Unary operators
2. Binary operators

1. **Unary Operators:**

   A unary operator operates only one operand. The Syntax of unary operator is given as follows*:*

   ```
   Operator operand
   ```

2. **Binary Operators:**

   A binary operator operates two operands. The Syntax of binary operator is given as follows **:**

   ```
   Operand1 operator operand2
   ```

There are various types of operators to cater to different purpose such as the following:

- Arithmetic operators
- Comparison operators
- Logical operators
- Set operators
- Concatenates operator

Arithmetic Operators: Arithmetic operators manipulate two operands and produce one result (Table 1.5). These operators are addition, subtraction, division, multiplication and Mod. These operators work on numeric data type for any calculation and addition, subtraction, and division operators also works on date data type.

Table 1.5 *Arithmetic Operators*

Operator	Description	Example	Result
/	Division	Select 345 / 4 from dual ;	86.25
*	Multiplication	Select 345 * 4 from dual ;	1380
+	Addition	Select 345 + 4 from dual ;	349
-	Subtraction	Select 345 - 4 from dual ;	341
Mod	Modulus (returns the reminder of m divided by n)	Select 345 mod 4 from dual ;	1

Dual Table: Dual is a dummy table in Oracle that could be used to perform temporary calculations and to check the result of any Oracle function on data which is not stored in any table. A dual table is consisting of only one row and a column.

Comparison Operators: Comparison operators are used to compare one expression with another (Table 1.6). The result of a comparison could be `TRUE, FALSE`. It is mainly used with `WHERE` clause of select, update and delete commands.

Table 1.6 *Comparison Operators*

Operater	Description	Example
=	Equal to	Select roll_no, name from student where c_code = 'PG001';
!= or <>	Not equal to	Select roll_no, name from student where c_code <> 'UG003';
<	Less than	Select c_name, duration from course where fee < 50000;
>	Greater than	Select c_name, duration from course where fee > 50000;
<=	Less than or equal to	Select c_name, duration from course where fee <= 45000;
>=	Greater than or equal to	Select c_name, duration from course where fee >= 56000;
In / Not In	Compare if a value lies within a specified list of values	Select * from student where name IN ('smith', 'john'); Select * from student where name IN NOT ('smith', 'john') ;
Between/ Not Between	Compare if a value lies within a specified range of values	Select c_name , duration from course where fee between 45000 and 56000;
Like / Not Like	Pattern matching	Select * from student where name LIKE 'd%;
Is Null / Is Not Null	Compare if a value is null	Select * from student where contact_no IS NULL;

Note: In a pattern matching operator `LIKE`, the underscore character (_) represents any one character and the percent character (%) represents a group of characters.

Logical Operators

Logical operators test for the truth of some condition (Table 1.7). Logical operators like comparison operators, return a Boolean data type with a value of `TRUE FALSE`, or `UNKNOWN`.

NOTES

Table 1.7 Logical Operators

Operator	Description	Example
AND	Returns TRUE if both conditions are TRUE. Returns FALSE if either is FALSE	Select c_name, duration from course where fee >= 45000 AND fee<= 56000 ;
OR	Returns TRUE if either condition is TRUE. Returns FALSE if both are FALSE.	Select roll_no, name from student where c_code = 'PG001' OR c_code = 'PG002' ;
NOT	Returns TRUE if the condition is FALSE. Returns FALSE if it is TRUE.	Select * from course where fee not > 78000 ;

Set Operators: Set operators combine the results of two queries into a single result (Table 1.8).

Table 1.8 Set Operators

Operator	Description	Example
UNION	Returns all distinct rows selected by either query.	Select roll_no, b_code from issue UNION Select roll_no, b_code from return ;
INTERSECT	Returns all distinct rows selected by both queries.	Select roll_no, b_code from issue INTERSECT Select roll_no, b_code from return ;
MINUS	Returns all distinct rows selected by the first query but not the second.	Select roll_no, b_code from issue MINUS Select roll_no, b_code from return ;

Concatenates Operator: Concatenates operator is used to concatenate two strings (Table 1.9).

Table 1.9 Concatenates Operators

Operator	Description	Example
||	Concatenates character strings	Select 'Student Name' || name from student ;

Creating a Table

This is a Data Definition Language (DDL) command that is used to define the structure of a table. In a table structure, you define various fields, their data types and constraints as per the requirement.

The syntax is as follows:

```
Create table < table_name >
( column_name  data type ( size ), column_name data type
( size ), …) ;
```

Example 1.6:

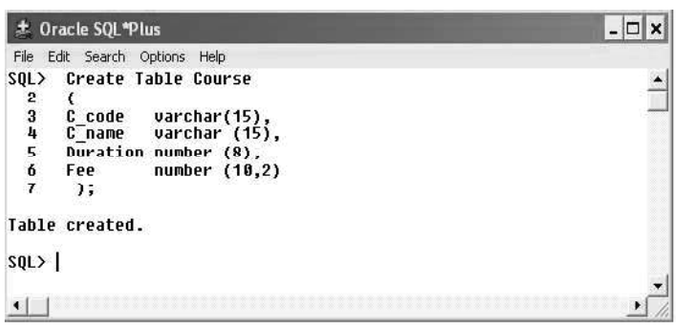
1. Create a table Course:

Column Name	Data Type	Size
c_code	varchar2	15
c_name	varchar2	15
duration	number	8
fee	number	10,2

The SQL command to create the table is as follows :

```
Create Table Course
(
C_code   varchar (15),  C_name      varchar (15),
Duration number (8),    Fee         number (10, 2)
) ;
```

This command will create table course and Oracle will prompt a message as shown here:



Example 1.7:

1. Create a table Student:

Column Name	Data Type	Size
Roll_No	Varchar	10
Name	Varchar	10
Address	Varchar	35
C_Code	Varchar	8

NOTES

The SQL command to create table is as follows:

```
Create Table Student
   (Roll_No          varchar (10),
   Name              varchar (10),
   Address           varchar (35),
   C_Code            varchar (8)
   ) ;
```

The above command will create a table structure to store student's information. Where `roll_no`, `name`, `address` and `c_code` are the field names and varchar is a data type.

Naming Convention:

The naming convention for tables is as follows:

- Every table must have the unique names.
- Table name should start with as alphabet.
- Only _ (under score) symbol could be used as separator in table name. No other special symbol could be used.
- The length of table name should not exceed from 256 characters.

Describing the Table Structure

Once a table is created, its structure could seen by giving Describe command.

The syntax is as follows:

```
Describe < table_name >
Or
Desc < table_name >
```

This command will describe the structure of course table as shown below here:

Name	Null ?	Type
C_CODE		ARCHAR2 (15)
C_NAME		VARCHAR2 (15)
DURATION		NUMBER (8)
FEE		NUMBER (10, 2)

Inserting Records in a Table

Once the structure of a table is created, the next action is to insert records on table. Insert is a Data Manipulation Language (DML) command.

The syntax is as follows:

```
Insert into < table name > values ( value1, value2 ,
...) ;
```

Example 1.8:

To insert records data into Course table, the command is as follows:

```
SQL > insert into course values ( 'PG001', 'MCA' , 3 ,
32000.00 ) ;
```

After executing this command, the system will prompt a message **1 row created**.

Insert few more records in the table:

```
SQL> insert into course values ( 'PG001', 'MCA' , 3 ,
32000.00 ) ;
SQL> insert into course values ( 'PG002, 'M Tech-CS' , 4
, 60000.00 ) ;
SQL> insert into course values ( 'PG004, 'M Tech-EC' , 4
, 64000.00 ) ;
```

Note: All char, varchar and date values should be enclosed in single quotes, for example, 'MCA' , '07-Sept-09', 'A-08-02', … .

Inserting Data into Specific Fields

With the above syntax of insert command it is necessary to insert data in all the fields in the same sequence as defined in the table. But sometimes few fields are required to update later on for example student's subjects marks are inserted in the table and total, percentage or grade is required to calculate later on. To deal such a situation you could you the following syntax:

The syntax to insert data into selected fields only:

```
Insert into < table name > ( column1, column2, …)
```

values (value1, value2, …) ;

The example to insert data into selected fields only:

```
insert into student ( roll_no, name, address )
values ( 'A-08-20','John','delhi' ) ;
```

Inserting Data with User Interaction

If hundreds or thousands of records are to be inserted in a table, it will be very tedious job to do it with the constant values. The other ways to insert records into table is take input from the user and repeat the command.

The Example to insert data with user interaction is as follows:

```
SQL > Insert into course values ('&C_code', '&C_name',
&duration, &fee ) ;
```

The Insert command with the `'&' operator` would ask for the input from the used as shown here:

```
Enter value for c_code : PG007
Enter value for c_name : M Sc-CS
Enter value for duration : 2
Enter value for fee : 32000
```

NOTES

After completion of the input system will prompt a message '1 row created. To insert more record, the same command could be repeated by putting / and pressing Enter key at SQL prompt.

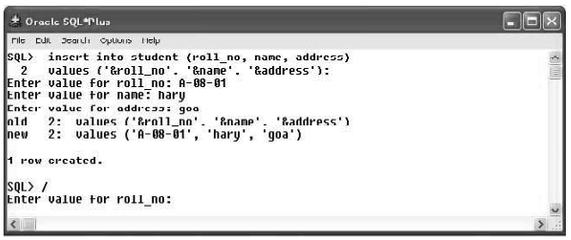
You could also insert records interactively into specific fields. The Example to insert data into selected fields with user's interaction as follows:

```
SQL> insert into student ( roll_no, name, address )
 values ( '&roll_no', '&name', '&address') ;
```

The insert command with the '&' operator would ask for the input from the user as shown here:

```
Enter value for roll_no : A-08-01
Enter value for name : hary
Enter value for address : goa
```

After completion of the input, system will prompt a message 1 row created. The screenshot for the same is given as follows:



Note: The '&' symbol would prompt user to input data to the various variable. The variable name that is written after '&' is not required to the same as field names.

Displaying Table Records

After inserting records into table, data could be displayed with the command select. All the fields and records could be displayed or only selective records and fields could be retrieved.

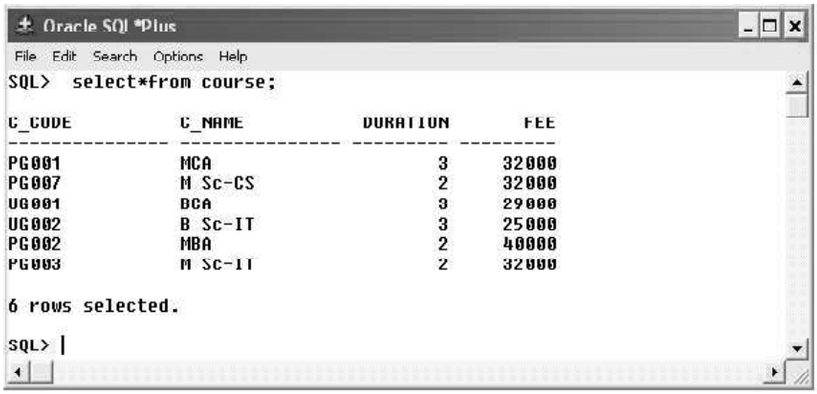
To retrieve all the columns of a table use '*' as shown here.

The syntax is as follows :

```
Select * from < table name > ;
```

The example is as follows:

```
Select * from course ;
```



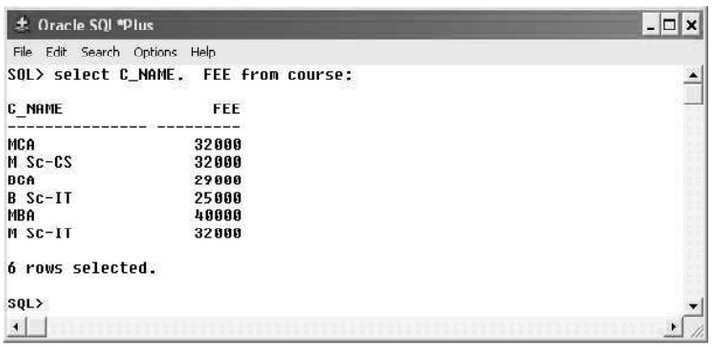
To view only selective fields enter column names separated by comma (,) as shown here.

The syntax is as follows:

```
Select field1, field2, … , from < table_names > ;
```

The example is as follows:

```
Select c_name,  fee from course ;
```



updating Table Records

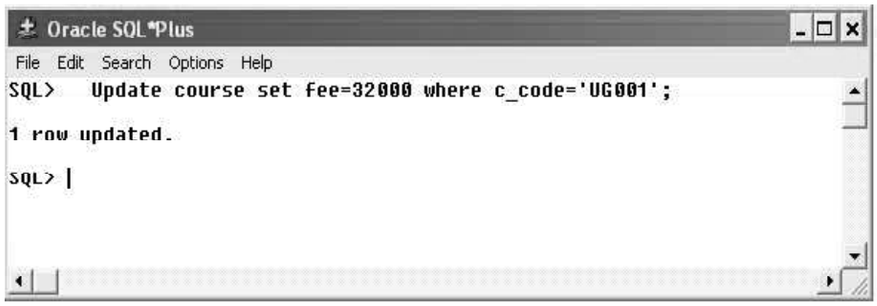
You may some times need to update the records that you have in your table. The son might be the contact no. of an address of any person has been changed or course fee is changed by the university. In such cases a Data Manipulation Language update command is used.

The syntax is as follows:

```
Update < table name >
Set < column_name1 = < new value > ,
     < column_name2 = < new value> ,
     …
     [ where < condition > ] ;
```

The example is as follows:

```
Update course set fee = 32000 where c_code = 'UG001' ;
```



This command will update the fee of course UG001 from Rs 29,000 to Rs 32,000.

The **Where** clause is used to specify the condition for which this fee should be changed. Without any condition all the records will be updated with the new fee Rs 32,000.

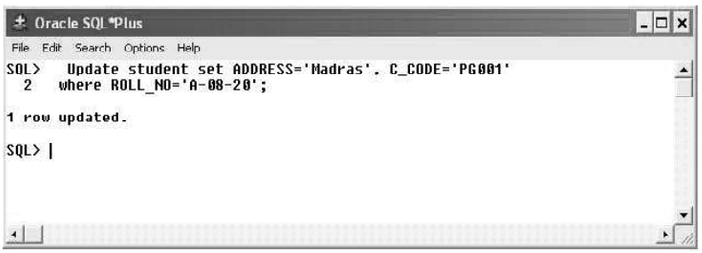
NOTES

NOTES

More than one columns could also be updated by specifying multiple columns and there new values after set key words.

The example is as follows:

```
Update student set ADDRESS = 'Madras', C_CODE = 'PG001'
 where ROLL_NO = 'A-08-20' ;
```



Deleting Records

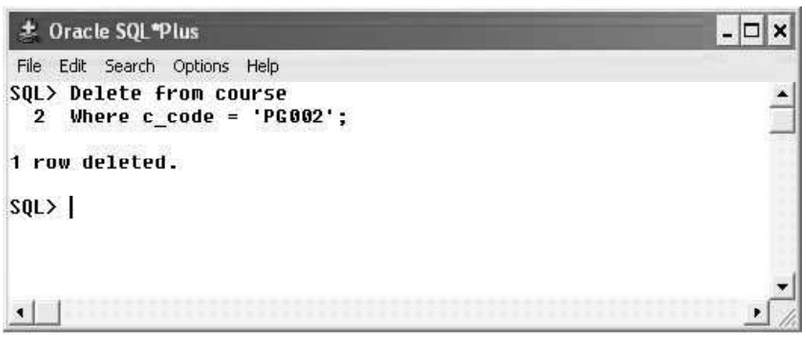
If records are no more needed you could delete records form the table. For this purpose you may use data manipulation (DML) command **Delete**. More than one, or all, records could be deleted from the table depending upon the Where condition.

The syntax is as follows:

```
Delete  < table_name >
[ where < condition > ] ;
      Or
Delete  from < table_name >
[ where < condition > ] ;
```

The example is as follows:

```
Delete from course where c_code = 'PG002' ;
```

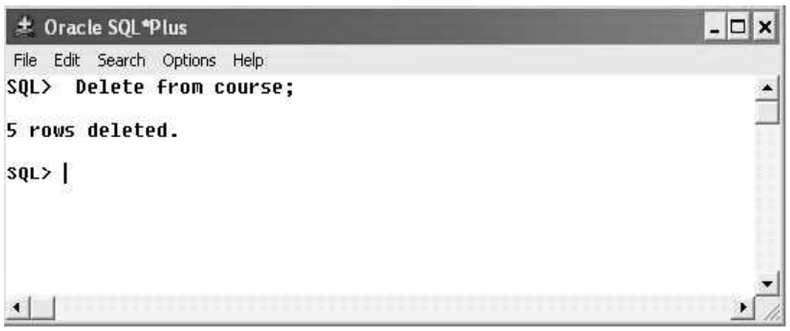


This command will delete one record from course table where course code is PG002. To delete all the records from a table, you could write the Delete command without Where clause as given here:

```
Delete from course ;
```

Or

```
Delete course ;
```



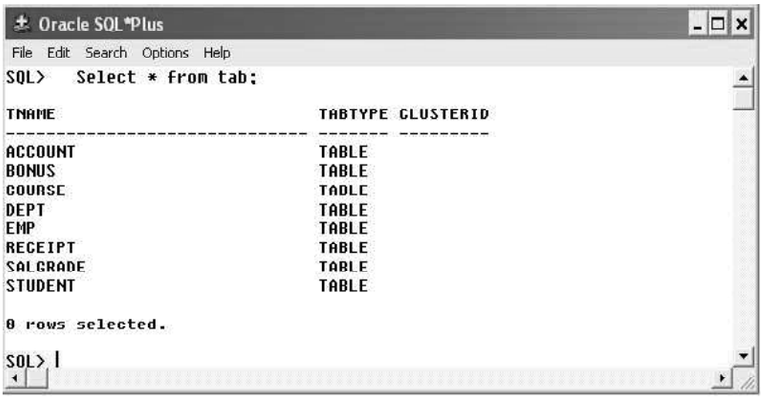
This command will delete all the five records from the course table.

Viewing the Existing Tables

To view all the existing tables in database, you could use `Tab`. Tab displays the name and type of object such as table, view, or synonym.

The example is as follows:

```
Select * from tab ;
```



`TNAME` is a column that displays the object name as table, view, index or synonym.

`TABTYPE` is a column which displays the type of object. The type of object may be any table, view, index, or synonym.

Filtering records using Where Conditions

A university could have thousand of records but all those records are not required for viewing every time. Many users might need to view different records from the same table at different time.

To filter various records of table, `Where` clause could be used with conditional, logical and other operators. Following are the examples of various operators in where clause of select query is as follows:

The syntax for select command with where clause :

```
Select * from < table name > [where < condition >] ;
```

Table 1.10 contains eight records. Let us filter records from this table with different conditions.

NOTES

Table 1.10 Filtering Records using where Clause

C_CODE	C_NAME	DURATION	FEE
PG001	MCA	3	55000
PG007	M Sc-CS	2	50000
UG001	BCA	3	32000
UG002	B Sc-IT	3	25000
PG003	M Sc-IT	2	48000
PG002	B Tech-CS	4	60000
PG004	B Tech-EC	4	64000
PG005	B Tech-IT	4	58000

Conditional Operators in SQL

Equal to (=)

To see the detail of course where course code equal to PG003 then the query will be

```
Select * from course where c_code = 'PG003' ;
```

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
PG003	M Sc-IT	2	48000

Not Equal to (<>, !=)

To see the detail of course where course duration is not 4 years then the query will be

```
Select * from course where duration <> 4 ;
```

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
PG001	MCA	3	55000
PG007	M Sc-CS	2	50000
UG001	BCA	3	32000
UG002	B Sc-IT	3	25000
PG003	M Sc-IT	2	48000

Greater Than (>)

To see the detail of course where course fee is greater than Rs 50,000 then the query will be

```
Select * from course where fee  > 50000 ;
```

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
PG001	MCA	3	55000
PG002	B Tech-CS	4	60000
PG004	B Tech-EC	4	64000
PG005	B Tech-IT	4	58000

As equal to, not equal to and greater than operators are used to filter records other operators as less than, less than equal to, greater than equal to could be used.

Other Operators in SQL:

BETWEEN

The `BETWEEN` operator filters the records between a given range. Suppose you want to filter the courses where fee is between Rs 45000 to Rs 58000. The query to retrieve such records is given as follows:

```
Select * from course  where fee between 45000 and 58000;
```

`Output` of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
PG001	MCA	3	55000
PG007	M Sc-CS	2	50000
PG003	M Sc-IT	2	48000
PG005	B Tech-IT	4	58000

The between operator can filter the numbers, text or date values.

NOT BETWEEN

The `NOT BETWEEN` operator filters the records where the data in not between a given range.

```
Select*from course where fee NOT BETWEEN 45000 and 58000
;
```

`Output` of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
UG001	BCA	3	32000
UG002	B Sc-IT	3	25000
PG002	B Tech-CS	4	60000
PG004	B Tech-EC	4	64000

IN

The operator could be used for char, varchar, date or number data type.

To see the detail of courses where course code is PG002, UG001 or PG004 the SQL query will be

```
Select * from course  where c_code in ('PG002', 'UG001',
'PG004' ) ;
```

NOTES

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
UG001	BCA	3	32000
PG002	B Tech-CS	4	60000
PG004	B Tech-EC	4	64000

NOT IN

```
Select * from course  where c_code NOT IN ( 'PG002',
'UG001', 'PG004' ) ;
```

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
PG001	MCA	3	55000
PG007	M Sc-CS	2	50000
UG002	B Sc-IT	3	25000
PG003	M Sc-IT	2	48000
PG005	B Tech-IT	4	58000

LIKE

The LIKE operator is used to filter records for a specific pattern and used only with CHAR and VARCHAR data types. When equal (=) operator matches the whole word, like operators allows to search the records with a particular pattern.

Like operator supports two wild card characters % (per cent) and _ (underscore).

- % (percent) represents the group of characters
- _ (underscore) represents one character

Like'%'

If you want to search the list of students whose first name starts with letter 'D', you could use the like operators in this way:

```
Select * from student where name like 'D%' ;
```

In this example % (per cent) represents the group of any characters (one or more). The '%' sign could be used both before and after the pattern. Following are the few examples of % (per cent) operators:

- 'Pt%' – search for the pattern where a string begins with the letters 'Pt'
- '%CS' - search for the pattern where a string ends with the 'CS'
- '%IT%' - search for the pattern where the string contains 'IT' anywhere

The example for Like '%' is given as follows:

```
Select * from course where c_name like 'B%' ;
```

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
UG001	BCA	3	32000
UG002	B Sc-IT	3	25000
PG002	B Tech-CS	4	60000
PG004	B Tech-EC	4	64000
PG005	B Tech-IT	4	58000

This command will retrieve courses where course name starts with 'B' letters and rest letter may be any one.

Like '_'

The underscore (_) operator is used to match a single character in a string pattern. If you want to search the list of courses where course name is of three characters only the SQL command will be as shown as follows:

```
Select * from course where c_name like '___' ;
```

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
PG001	MCA	3	55000
UG001	BCA	3	32000

NOT LIKE

NOT LIKE is reverse of LIKE which filters the records where the given pattern does not match.

```
Select * from course where c_name NOT LIKE '___'  ;
```

Output of this query is shown as follows:

C_CODE	C_NAME	DURATION	FEE
PG007	M Sc-CS	2	50000
UG002	B Sc-IT	3	25000
PG003	M Sc-IT	2	48000
PG002	B Tech-CS	4	60000
PG004	B Tech-EC	4	64000
PG005	B Tech-IT	4	58000

IS NULL

Oracle treats the missing values as NULL values in database. NULL values represent missing unknown data in a column which is different from 0 or space. To filter the NULL values IS NULL operator is used as shown as follows:

ROLL_NO	NAME	ADDRESS	C_CODE
A-08-20	John	Delhi	
A-08-01	Hary	Goa	
A-08-02	Rohan	Bangalore	UG002
A-08-04	Jone	Pune	PG005
B-08-01	Bandi cos	Pune	PG005
B-08-02	Ravat	Pune	PG006

NOTES

To see the students, where course code is unknown the query is as follows:

```
Select * from student where c_code IS NULL ;
```

Output of this query is shown as follows:

ROLL_NO	NAME	ADDRESS	C_CODE
A-08-20	John	delhi	
A-08-01	hary	goa	

IS NOT NULL

`IS NOT NULL` is reverse of `NOT NULL`. It filters the records where the specified column values are not null.

To see the students where course code is known the query is as follows:

```
Select * from student where c_code is not null ;
```

Output of this query is shown as follows:

ROLL_NO	NAME	ADDRESS	C_CODE
A-08-02	rohan	bangalore	UG002
A-08-04	jone	pune	PG005
B-08-01	bandi cos	pune	PG005
B-08-02	ravat	pune	PG006

This command returns where student course code is not null.

Logical Conditions in SQL:

AND

The AND filters the records if all the given conditions are true.

Example: 1.9

```
Select ROLL_NO, NAME from student where address = 'pune'
AND c_code = 'PG005' ;
```

Output of this query is shown as follows:

ROLL_NO	NAME
A-08-04	jone
B-08-01	bandi cos

OR

The OR filters the records if either condition is true.

Example:

```
Select ROLL_NO, NAME from student where address = 'pune'
OR c_code = 'PG005' ;
```

Output of this query is shown as follows:

ROLL_NO	NAME
A-08-04	jone
B-08-01	bandi cos
B-08-02	ravat

NOT

Filters the records where specified condition is false.

Example 1.10:

```
Select * from student where  NOT c_code = 'PG005' ;
```

Output of this query is shown as follows:

ROLL_NO	NAME	ADDRESS	C_CODE
A-08-02	rohan	bangalore	UG002
B-08-02	ravat	pune	PG006

Display Distinct Values:

In a table various duplicate values could be stored. But sometimes it is required to display only distinct records. To display distinct cities of students where they belong to, the SQL command is as follows :

```
Select DISTINCT address from student ;
```

Output of this query is shown as follows:

ADDRESS
Bangalore
Delhi
goa
pune

Modify Table Structure:

Once a table structure is created, its structure could be modified as per the requirements of the organization.

Various types of modification may be done such as the following:

- Add a new column or constraint
- Delete any exiting column or constraint
- Modify the data type and size of data type

SQL provide an `ALTER` command to modify a table structure. It is a Data Definition Language (DDL) command. Following are the few examples to modify a table structure.

Add a New Column:

The syntax for alter command:

```
Alter table < table_name >
ADD ( column_name  data type ( length ) , column_name
data type ( length ) , …) ;
```

The example for alter command:

Alter table student add (mobile Number (10)) ;

NOTES

NOTES

This command will add a new column mobile in student table. You could see the new structure of student table:

```
desc student ;
```

Change data type of an existing column :

The syntax for alter command :

```
Alter table < table_name > modify ( column data type (
length ) ,
```

column data type (length) ,…) ;

The example for alter command:

```
Alter table course modify c_code char ( 15 ) ;
```

This command will change the data type of c_code field from varchar to char.

Modify the length of on existing column:

The syntax for alter command :

```
Alter table < table_name > modify ( column data type (
length ) ,
```

column data type (length) , …) ;

The example for alter command :

Alter table student modify (name varchar (20) , address Varchar2 (40)) ;

This command will change the length of name column from 15 to 20 and address from 35 to 40.

Important points to remember

- If table column contains the values, then the length of column could be increase.
- To change the data type column should be empty.
- To decrease the size of data type column should be empty.

Delete any Column:

The syntax for alter command :

```
Alter table < table name > drop column column_name ;
```

The example for alter command :

```
Alter table student drop column mobile ;
```

The above command will delete the column mobile form students table.

The syntax for alter command :

```
Alter table < table_name > modify ( column data type (
length ) ,
column data type ( length ) , … ) ;
```

The example for alter command :

```
Alter table student modify ( name varchar (20), address
Varchar2 (40) ) ;
```

NOTES

Renaming Tables

To rename table, you could use Rename command.

The syntax for alter command:

```
Rename old_table_name to new_table_name ;
```

The example for alter command:

```
Rename student to student_MCA ;
```

Remove Table:

When a SQL table is no more required, you could get rid of the table by using DROP command. Drop table is a Data Definition Language (DDL). Drop command is used to drop any object such as table, index, view, package and function.

The syntax for drop table:

```
Drop table < table_name >
```

The example for drop table:

```
Drop table Course;
```

The above command will remove the course tables.

1.6 ORACLE FUNCTIONS

SQL functions are built-in functions used for different purposes such as calculation, comparison and conversion of data. Functions may or may not have the arguments (input) and have the capability to return a value.

Oracle provided various built-in functions for different purpose such as calculation, comparison and conversion of data. Functions may or may not have the arguments (input) and have the capability to return a value.

Types of Functions

Basically there are two types of functions:

- Aggregate functions
- Scalar functions

Aggregate functions

Aggregate functions work on a group of values (column values) and return a single value. Few aggregate functions are listed:

- SUM()
- MAX()
- MIN()
- AVG()
- COUNT()

Scalar functions

SQL scalar functions return a single value, based on the input value. Few scalar functions are listed:

NOTES

- MID()
- LEN()
- Upper()
- Lower()

Let suppose we have a table Order with the following records:

Order_ID	Item_ID	Order_Date	Price
20040	I 001	10- Sept - 2009	6700
20040	I 004	10- Sept - 2009	600
20041	I 001	23- Sept - 2009	6700
20041	I 003	23- Sept - 2009	4560
20042	I 003	03- Oct - 2009	4560
20043	I 005	07- Oct - 2009	380

Table - Order

Sum ():

To see the sum of price of the item_ID I003 SQL query is as follows:

```
Select sum ( price ) from order where Item_ID = 'I003';
```

The output of the above query is 23500.

Min ()

To see the order detail where item price is minimum, SQL query is as follows:

```
Select  min ( price ) from order ;
```

The output of the above query is 380.

Max ()

To see the order detail where item price is maximum SQL query is as follows:

```
Select  max ( price ) from order ;
```

The output of the above query is 6700.

Count ()

To see the number of orders for item_ID I001 SQL query is as follows :

```
Select  count ( item_id ) from order where item_ID =
'I001' ;
```

The output of the above query is 2.

Count (*)

To see the number of records in a table SQL query is as follows:

```
Select count (*)  from order ;
```

The **output** of the above query is 8.

Upper () :

To convert the text to uppercase SQL query is as follows:

```
Select upper ('Computer') from dual ;
```

The output of the above query is COMPUTER.

Lower () :

To converts the text to upper case SQL query is as follows:

```
Select lower ('Computer') from dual ;
```

The output of the above query is computer.

Round (n)

To round of any number SQL query is as follows:

```
Select round ( 1.23456, 2 ) from dual;
```

The output of the above query is 1.23.

Sqrt (n)

It calculates square root value of number SQL query is as follows:

```
Select sqrt ( 49 ) from dual ;
```

The output of the above query is 7.

1.7 INTRODUCTION TO SYNONYMS

A synonym is an elective name for articles like tables, sees, arrangements, put away methods, and other data set items. The most part use equivalents when you are allowing admittance to an article from one more blueprint and you don't need the clients to need to stress over knowing which diagram claims the item.

Create Synonym

You might wish to make an equivalent so clients don't need to prefix the table name with the mapping name when utilizing the table in a question.

Syntax

The grammar to make an equivalent word in Oracle is:

CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema.] synonym_name

FOR [schema.] object_name [@ dblink];

OR replace can be defined as:

Permits you to reproduce the equivalent (assuming it as of now exists) without giving a DROP equivalent order.

Public Keyword

It implies that the equivalent is a public equivalent and is open to all clients. Recall however that the client should initially have the proper advantages to the item to utilize the equivalent word.

NOTES

Schema

The suitable outline. On the off chance that this expression is precluded, Oracle expects that you are alluding to your own blueprint.

object_name

The name of the item for which you are making the equivalent word. It tends to be one of the accompanying:

For instance:

CREATE PUBLIC SYNONYM suppliers FOR app.suppliers;

This initially CREATE SYNONYM model shows how to make an equivalent called providers. Presently, clients of different patterns can reference the table called providers without prefixing the table name with the mapping named application. For instance:

SELECT * FROM suppliers;

To reclassify it, you could generally utilize the OR REPLACE express as follows:

CREATE OR REPLACE PUBLIC SYNONYM suppliers FOR app.suppliers;

Drop synonym

When an equivalent has been made in Oracle, you may eventually have to drop the equivalent word.

Syntax

The punctuation to drop an equivalent in Oracle is:

DROP [PUBLIC] SYNONYM [schema.] synonym_name [force];

PUBLIC

Permits you to drop a public equivalent word. On the off chance that you have determined PUBLIC, then, at that point, you don't indicate a pattern.

Force

It will drive Oracle to drop the equivalent word regardless of whether it has conditions. It is likely not a smart thought to utilize power as it can cause nullification of Oracle objects.

For instance

How about we take a gander at an illustration of how to drop an equivalent in Oracle.

For example:

DROP PUBLIC SYNONYM suppliers;

This DROP explanation would drop the equivalent word called providers that we characterized before.

1.8 INTRODUCTION TO SEQUENCES

NOTES

The make Sequence proclamation is utilized to make an arrangement, which is an information base item from which various clients might produce novel whole numbers. You can utilize arrangements to naturally produce essential key qualities.

At the point when an arrangement number is created, the succession is augmented, autonomous of the exchange submitting or moving back. Assuming two clients simultaneously increase a similar arrangement, the grouping numbers every client secures may have holes, since succession numbers are being created by the other client. One client can never obtain the succession number created by another client. Later a succession esteem is produced by one client, that client can keep on getting to that esteem whether or not the grouping is augmented by another client.

Grouping numbers are created autonomously of tables, so a similar arrangement can be utilized for one or for a long time. It is conceivable that singular arrangement numbers will seem, by all accounts, to be skipped, in light of the fact that they were created and utilized in an exchange that eventually moved back. Furthermore, a solitary client may not understand that different clients are drawing from a similar succession.

Later an arrangement is made, you can get to its qualities in SQL articulations with the CURRVAL pseudocolumn, which returns the current worth of the succession, or the NEXTVAL pseudocolumn, which augments the grouping and returns the new worth.

Assuming you indicate none of the accompanying provisions, then, at that point, you make a climbing arrangement that beginnings with 1 and increments by 1 with no maximum cutoff. Indicating just INCREMENT BY - 1 makes a diving succession that beginnings with - 1 and diminishes with no lower limit.

To make an arrangement that additions without headed, for rising groupings, discard the MAXVALUE boundary or indicate NOMAXVALUE. For slipping groupings, exclude the MINVALUE boundary or determine the NOMINVALUE.

To make an arrangement that stops at a predefined limit, for a rising grouping, determine an incentive for the MAXVALUE boundary. For a slipping grouping, determine an incentive for the MINVALUE boundary. Additionally determine NOCYCLE. Any endeavor to create a grouping number once the arrangement has arrived at its breaking point brings about a mistake.

To make an arrangement that restarts in the wake of coming to a predefined limit, indicate values for both the MAXVALUE and MINVALUE boundaries. Likewise indicate CYCLE.

INCREMENT BY

Determine the span between arrangement numbers. This whole number worth can be any certain or negative whole number, yet it can't be 0. This worth can have 28

NOTES

or less digits for a rising arrangement and 27 or less digits for a plunging succession. The outright of this worth should be not exactly the distinction of MAXVALUE and MINVALUE. Assuming this worth is negative, then, at that point, the arrangement plunges. Assuming the worth is positive, then, at that point, the grouping climbs. In the event that you exclude this condition, then, at that point, the stretch defaults to 1.

START WITH

Determine the primary grouping number to be created. Utilize this provision to begin a climbing arrangement at a worth more noteworthy than its base or to begin a plunging succession at a worth not as much as it's greatest. For climbing arrangements, the default esteem is the base worth of the succession. For diving arrangements, the default esteem is the greatest worth of the grouping. This number worth can have 28 or less digits for positive qualities and 27 or less digits for negative qualities.

MAXVALUE

Indicate the greatest worth the succession can create. This number worth can have 28 or less digits for positive qualities and 27 or less digits for negative qualities. MAXVALUE should be equivalent to or more prominent than START WITH and should be more noteworthy than MINVALUE.

NOMAXVALUE

Determine NOMAXVALUE to show a most extreme worth of 1028-1 for a rising arrangement or - 1 for a dropping succession. This is the default.

MINVALUE

Indicate the base worth of the grouping. This number worth can have 28 or less digits for positive qualities and 27 or less digits for negative qualities. MINVALUE should be not exactly or equivalent to START WITH and should be not as much as MAXVALUE.

NOMINVALUE

Determine NOMINVALUE to show a base worth of 1 for a climbing arrangement or - (1027 - 1) for a slipping succession. This is the default.

CYCLE

Determine CYCLE to demonstrate that the arrangement keeps on producing values in the wake of arriving at either its most extreme or least worth. Later a climbing grouping arrives at its greatest worth, it creates its base worth. Later a plunging grouping arrives at its base, it creates its most extreme worth.

NOCYCLE

Determine NOCYCLE to show that the grouping can't produce more qualities subsequent to arriving at its most extreme or least worth. This is the default.

NOTES

CACHE

Determine the number of upsides of the arrangement the information base preallocates and keeps in memory for quicker access. This number worth can have 28 or less digits. The base incentive for this boundary is 2. For successions that cycle, this worth should be not exactly the quantity of qualities in the cycle. You can't store a bigger number of qualities than will fit in a given pattern of succession numbers.

NOCACHE

Determine NOCACHE to show that upsides of the arrangement are not preallocated. On the off chance that you preclude both CACHE and NOCACHE, then, at that point, the information base reserves 20 arrangement numbers of course.

ORDER

Determine ORDER to ensure that succession numbers are created arranged according to popular demand. This condition is valuable assuming you are utilizing the succession numbers as timestamps. Ensuring request is normally not significant for successions used to create essential keys.

Request is fundamental just to ensure requested age on the off chance that you are utilizing Oracle Real Application Clusters. In the event that you are utilizing selective mode, then, at that point, succession numbers are constantly produced all together.

NOORDER

Indicate NOORDER to ensure arrangement numbers are produced arranged according to popular demand. This is the default.

KEEP

Indicate KEEP assuming that you need NEXTVAL to hold its unique worth during replay for Application Continuity. This conduct will happen provided that the client running the application is the proprietor of the mapping containing the arrangement. This provision is helpful for giving tie variable consistency at replay later recoverable mistakes. Allude to Oracle Database Development Guide for more data on Application Continuity.

NOKEEP

Indicate NOKEEP assuming you don't need NEXTVAL to hold its unique worth during replay for Application Continuity. This is the default.

Example:

```
CREATE SEQUENCE customers_seq
 START WITH     1000
 INCREMENT BY   1
 NOCACHE
```

NOCYCLE;

1.8.1 Alternating of Sequence

NOTES

You typically view your expense code structures by their alphanumeric expense code request. Be that as it may, you can organize and see your expense code structures dependent on different successions you pick. You can relegate substitute successions for various reasons, including:

To recognize explicit spaces of a task

To observe the rules of administrative expense code structures

To agree with the prerequisites of a parent organization

To agree with the prerequisites of an outsider

You can allocate substitute successions with client characterized class codes that immediate the framework to modify your expense code structure dependent on the classification codes. You can likewise enter another expense code number for every one of the records in the expense code structure.

You can utilize the accompanying strategies to empower your framework to perceive substitute successions for your expense code structures:

You can dole out substitute succession classification codes or potentially substitute arrangement cost code numbers physically to each line of an expense code structure later you make it. You can guide your framework to dole out substitute succession classification codes consequently when you make another expense code structure. You can utilize a worldwide update program to naturally allot substitute succession class codes or substitute expense code numbers to a whole expense code structure later you make it.

1.9 INTRODUCTION TO INDEXES

In the event that a list has been set up and the SQL articulation is arrangement to exploit the record, the list will be looked through first and the entrance time will most likely be discernibly lower assuming you are managing a huge data set. Without a list Oracle does a full table inquiry looking at each column. Prophet is effectively engaged with utilizing lists to fulfill questions and inspects the inquiry to figure out what files it will utilize.

In Oracle, records are gathered with the idea of imperatives. There are two significant classifications of limitations: respectability requirements that allude to the critical fields and worth imperatives that arrangement with information went into a segment. A requirement is utilized to ensure the legitimacy of information in one or various tables and forestall invalid sections. In particular, requirements authorize specific standards managing a table or a section of that table and can be utilized to forestall the cancellation of a table that has youngsters or conditions. Lists as requirements are ensuring that the essential key field is one of a kind and that the association through an unfamiliar key is substantial.

The requirements that we will take a gander at are displayed in the table underneath.

Constraint	Processing
CHECK	Allows the specification of a condition on the data
FOREIGN KEY	Key used in the relationship of two tables
PRIMARY KEY	Unique key to each row in the table - uniquely identifies row
NOT NULL	Column must not be null
UNIQUE	Column(s) that must be unique for each row in the table

At the point when the developer is utilizing requirements, they have the choice of naming them (then, at that point, the name can be significant) or having the framework create a name with the SYS-Cn design. Requirements can be essential for the interaction to make a table or they should be possible as upkeep of the table. Since obliges can be on a segment or on a table they can be characterized at one or the other level. To see the imperatives that have been relegated to a specific table do a SELECT from the USER_CONSTRAINTS information word reference table.

To see all tables use:

SELECT * FROM USER_CONSTRAINTS;

To see a particular table, use:

SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = ‘Benefactor’;

In the model underneath, I have characterized one field as an essential key and put a really take a look at requirement on another.

SQL CODE:

```
1 CREATE TABLE TRYKEY1
2  (idno NUMBER(3) CONSTRAINT idno_pk PRIMARY KEY,
3  name VARCHAR2(20),
4* deptno NUMBER(2) CONSTRAINT valid_dept_ch CHECK (deptno > 0
AND deptno < 20))
SQL> /
Table created.
```

1.10 DATA INTEGRITY

Data genuinely should keep up with information uprightness, which is adherence to business rules controlled by the data set director or application designer. When planning an information base application, engineers have a few choices for ensuring the respectability of information put away in the data set.

These choices include:

NOTES

- Authorizing business rules with set off put away data set techniques.
- Utilizing put away strategies to totally control admittance to information.
- Authorizing business rules in the code of a data set application.
- Utilizing Oracle Database uprightness limitations, which are rules characterized at the section or item level that confine values in the data set.

The Purpose and Benefits of Integrity Constraints

A trustworthiness requirement is a blueprint object that is made and dropped utilizing SQL. To uphold information honesty, use uprightness limitations sooner rather than later.

Benefits of honesty imperatives over choices for authorizing information respectability include:

- Declarative ease

  Since you characterize honesty imperatives utilizing SQL proclamations, no extra writing computer programs is required when you characterize or modify a table. The SQL proclamations are not difficult to compose and dispense with programming blunders.

- Centralized rules

  Honesty imperatives are characterized for tables and are put away in the information word reference. Consequently, information entered by all applications should cling to similar honesty limitations. Assuming the principles change at the table level, then, at that point, applications need not change. Likewise, applications can utilize metadata in the information word reference to quickly illuminate clients regarding infringement, even before the data set really looks at the SQL proclamation.

- Flexibility when loading data

  You can handicap respectability requirements briefly to keep away from execution overhead when stacking a lot of information. At the point when the information load is finished, you can re-empower the respectability requirements.

Types of Integrity Constraints and Preregquisites

Prophet Database empowers you to apply imperatives both at the table and section level.

A requirement indicated as a component of the meaning of a section or quality is an inline detail. A requirement determined as a feature of the table definition is an off the mark.

A key is the segment or set of segments remembered for the meaning of particular kinds of trustworthiness requirements. Keys portray the connections between the tables and sections of a social data set. Individual qualities in a key are called key qualities.

The accompanying table depicts the kinds of limitations. Each can be indicated either inline or off the mark, aside from NOT NULL, which should be inline.

Table 1.11 displays the keywords and parameters of data integrity.

***Table 1.11**: Integrity Constraints*

Constraint Type	Description	See Also
NOT NULL	Allows or disallows inserts or updates of rows containing a **null** in a specified column.	"NOT NULL Integrity Constraints"
Unique key	Prohibits multiple rows from having the same value in the same column or combination of columns but allows some values to be null.	"Unique Constraints"
Primary key	Combines a NOT NULL constraint and a unique constraint. It prohibits multiple rows from having the same value in the same column or combination of columns and prohibits values from being null.	"Primary Key Constraints"
Foreign key	Designates a column as the foreign key and establishes a relationship between the foreign key and a primary or unique key, called the **referenced key.**	"Foreign Key Constraints"
Check	Requires a database value to obey a specified condition.	"Check Constraints"
REF	Dictates types of data manipulation allowed on values in a REF column and how these actions affect dependent values. In an object-relational database, a built-in data type called a REF encapsulates a reference to a row object of a specified object type. Referential integrity constraints on REF columns ensure that there is a row object for the REF.	*Oracle Database Object-Relational Developer's Guide* to learn about REF constraints

1.11 REFERENCE CLAUSE

Rules for REF segments and traits can be upheld by the utilization of imperatives.

In Oracle Database, a REF segment or trait can be unconstrained or obliged utilizing a SCOPE condition or a referential requirement statement. At the point when a REF segment is unconstrained, it might store object references to push objects contained in any item table of the relating object type.

Prophet Database doesn't guarantee that the item references put away in such segments highlight legitimate and existing line objects. In this way, REF sections might contain object references that don't highlight any current line object. Such REF esteems are alluded to as hanging references.

A SCOPE requirement can be applied to a particular article table. All the REF esteems put away in a section with a SCOPE limitation point at line objects of the table indicated in the SCOPE proviso. The REF esteems may, be that as it may, be hanging.

A REF section might be obliged with a REFERENTIAL imperative like the determination for unfamiliar keys. The standards for referential imperatives apply to such sections. That is, the item reference put away in these segments should highlight a legitimate and existing line object in the predefined object table.

Essential KEY imperatives can't be determined for REF segments. Notwithstanding, you can determine NOT NULL imperatives for such segments.

Name Resolution

There are multiple ways of settling names in Oracle Database.

Prophet SQL allows you to discard qualifying table names in some social activities.

NOTES

For instance, assuming dept_addr is a segment in the department_loc table and old_office is a section in the development table, you can utilize the accompanying:

SELECT * FROM department_loc WHERE EXISTS

(SELECT * FROM development WHERE dept_addr = old_office);

1.11.1 SCOPE REF Constraints

Utilize a requirement to characterize a respectability limitation—a standard that confines the qualities in a data set. Prophet Database allows you to make six sorts of limitations and allows you to pronounce them in two ways.

The six kinds of honesty requirement are depicted momentarily here and all the more completely in "Semantics ":

A NOT NULL limitation forbids a data set worth from being invalid. A special limitation disallows numerous lines from having similar worth in similar section or mix of segments yet permits a few qualities to be invalid. An essential key limitation consolidates a NOT NULL imperative and a novel requirement in a solitary revelation. That is, it forbids various lines from having similar worth in similar segment or blend of segments and restricts values from being invalid. An unfamiliar key limitation requires values in a single table to match esteems in another table.

A check limitation requires a worth in the information base to agree with a predefined condition.

A REF section by definition references an item in another article type or in a social table. A REF requirement allows you to additionally portray the connection between the REF section and the article it references.

You can characterize requirements grammatically in two ways:

As a feature of the meaning of a singular segment or characteristic. This is called inline determination.

As a feature of the table definition. This is called off the mark detail.

NOT NULL imperatives should be proclaimed inline. Any remaining limitations can be announced either inline or off the mark.

View Constraints

Prophet Database doesn't authorize view requirements. Nonetheless, you can implement requirements on sees through limitations on base tables.

You can determine just novel, essential key, and unfamiliar key limitations on perspectives, and they are upheld just in DISABLE NOVALIDATE mode. You can't characterize view requirements on characteristics of an item section.

NOT NULL Constraints

A NOT NULL requirement forbids a section from containing nulls. The NULL catchphrase without anyone else doesn't really characterize a respectability limitation, yet you can indicate it to expressly allow a section to contain nulls. You should characterize NOT NULL and NULL utilizing inline particular. Assuming

NOTES

you indicate neither NOT NULL nor NULL, then, at that point, the default is NULL.

NOT NULL limitations are the main imperatives you can indicate inline on XMLType and VARRAY segments.

To fulfill a NOT NULL requirement, each line in the table should contain an incentive for the section.

Unique Constraints

A special imperative assigns a section as an extraordinary key. A composite remarkable key assigns a mix of sections as the one of a kind key. At the point when you characterize a remarkable requirement inline, you really want just the UNIQUE catchphrase. At the point when you characterize a novel imperative off the mark, you should likewise determine at least one segments. You should characterize a composite novel key off the mark.

To fulfill an interesting imperative, no two columns in the table can have a similar incentive for the remarkable key. Notwithstanding, the novel key comprised of a solitary segment can contain nulls. To fulfill a composite extraordinary key, no two lines in the table or view can have similar mix of qualities in the key segments. Any line that contains nulls in all key segments naturally fulfills the imperative. Nonetheless, two lines that contain nulls for at least one critical sections and similar blend of qualities for the other key segments disregard the imperative.

At the point when you indicate a one of a kind requirement on at least one segments, Oracle verifiably makes a file on the exceptional key. Assuming you are characterizing uniqueness for reasons for question execution, then, at that point, Oracle suggests that you rather make the remarkable file unequivocally utilizing a CREATE UNIQUE INDEX proclamation. See CREATE INDEX for more data.

Limitations on Unique Constraints

None of the segments in the special key can be of LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, OBJECT, REF, TIMESTAMP WITH TIME ZONE, or client characterized type. Notwithstanding, the special key can contain a segment of TIMESTAMP WITH LOCAL TIME ZONE.

A composite remarkable key can't have in excess of 32 sections.

You can't assign similar section or blend of segments as both an essential key and a novel key.

You can't determine a remarkable key while making a subview in a legacy progressive system. The extraordinary key can be indicated uniquely for the high level (root) view.

Primary Key Constraints

An essential key requirement assigns a segment as the essential key of a table or view. A composite essential key assigns a mix of segments as the essential key. At the point when you characterize an essential key requirement inline, you really want just the PRIMARY KEY catchphrases. At the point when you characterize an essential key imperative off the mark, you should likewise determine at least one sections. You should characterize a composite essential key off the mark.

NOTES

An essential key requirement consolidates a NOT NULL and interesting limitation in one assertion. In this manner, to fulfill an essential key requirement:

No essential key worth can show up in more than one line in the table.

No segment that is essential for the essential key can contain an invalid.

Limitations on Primary Key Constraints

A table or view can have just a single essential key.

None of the segments in the essential key can be LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, BFILE, REF, TIMESTAMP WITH TIME ZONE, or client characterized type. Notwithstanding, the essential key can contain a section of TIMESTAMP WITH LOCAL TIME ZONE.

The size of the essential key can't surpass roughly one information base square.

A composite essential key can't have in excess of 32 sections.

You can't assign similar segment or mix of segments as both an essential key and a one of a kind key.

You can't determine an essential key while making a subview in a legacy order. The essential key can be determined distinctly for the high level (root) view.

REF Constraints

REF imperatives let you portray the connection between a segment of type REF and the article it references.

ref_constraint

REF imperatives utilize the ref_constraint punctuation. You characterize a REF limitation either inline or off the mark. Off the mark detail expects you to determine the REF segment or characteristic you are further portraying.

For ref_column, indicate the name of a REF section of an article or social table.

For ref_attribute, determine an implanted REF trait inside an article segment of a social table.

Both inline and off the mark particular let you characterize an extension limitation, a rowid imperative, or a referential trustworthiness requirement on a REF section.

On the off chance that the degree table or referred to table of the REF section has an essential key-based article identifier, then, at that point, the REF segment is a client characterized REF segment.

SCOPE REF Constraints

In a table with a REF section, every REF esteem in the segment might possibly reference a line in an alternate article table. The SCOPE condition confines the extent of references to a solitary table, scope_table. The qualities in the REF section or property highlight objects in scope_table, in which item occurrences of a similar kind as the REF segment are put away.

NOTES

Indicate the SCOPE condition to confine the extent of references in the REF section to a solitary table. For you to indicate this condition, scope_table should be in your own outline or you should have SELECT advantages on scope_table or SELECT ANY TABLE framework advantages. You can determine just a single degree table for every REF section.

Limitations on Scope Constraints

You can't add a degree limitation to a current segment except if the table is vacant.

You can't indicate a degree imperative for the REF components of a VARRAY section.

You should indicate this proviso in the event that you determine AS subquery and the subquery returns client characterized REFs.

You can't in this way drop a degree imperative from a REF section.

1.11.2 VALIDATE

The conduct of VALIDATE and NOVALIDATE consistently relies upon whether the requirement is empowered or impaired, either unequivocally or as a matter of course.

Along these lines they are depicted with regards to "Enable Clause" and "Disable Clause".

ENABLE Clause

Determine ENABLE assuming you need the limitation to be applied to the information in the table.

Assuming that you empower an exceptional or essential key limitation, and in the event that no record exists on the key, then, at that point, Oracle Database makes a remarkable file. Except if you determine KEEP INDEX when therefore handicapping the requirement, this list is dropped and the information base revamps the record each time the limitation is reenabled.

You can likewise try not to modify the list and dispose of repetitive records by making new essential key and exceptional limitations at first crippled. Then, at that point, make (or utilize existing) nonunique records to authorize the imperative. Prophet doesn't drop a nonunique file when the requirement is incapacitated, so resulting ENABLE tasks are worked with.

Empower VALIDATE indicates that all old and new information additionally follows the imperative. An empowered approved limitation ensures that all information is and will keep on being substantial.

On the off chance that any column in the table disregards the respectability limitation, the imperative remaining parts impaired and Oracle returns a blunder. Assuming all lines agree with the requirement, Oracle empowers the limitation. In this way, on the off chance that new information abuses the limitation, Oracle doesn't execute the assertion and returns a mistake demonstrating the uprightness requirement infringement.

Assuming you place an essential key requirement in ENABLE VALIDATE mode, the approval interaction will confirm that the essential key sections contain

NOTES

no nulls. To keep away from this overhead, mark every segment in the essential key NOT NULL prior to entering information into the segment and prior to empowering the essential key imperative of the table.

Empower NOVALIDATE guarantees that all new DML procedure on the obliged information conform to the limitation. This proviso doesn't guarantee that current information in the table conforms to the imperative and hence doesn't need a table lock.

In the event that you determine neither VALIDATE nor NOVALIDATE, the default is VALIDATE.

On the off chance that you change the condition of any single limitation from ENABLE NOVALIDATE to ENABLE VALIDATE, the activity can be acted in equal, and doesn't obstruct peruses, composes, or other DDL tasks.

Restriction on the ENABLE Clause

You can't empower an unfamiliar key that references a handicapped one of a kind or essential key.

DISABLE Clause

Indicate DISABLE to debilitate the respectability limitation. Impaired trustworthiness imperatives show up in the information word reference alongside empowered requirements. Assuming that you don't indicate this provision while making a limitation, Oracle naturally empowers the requirement.

Debilitate VALIDATE incapacitates the imperative and drops the file on the limitation, yet keeps the requirement legitimate. This element is generally helpful in information warehousing circumstances, since it allows you to stack a lot of information while likewise saving space by not having a record. This setting allows you to stack information from a nonpartitioned table into a divided table utilizing the exchange_partition_clause of the ALTER TABLE explanation or utilizing SQL*Loader. Any remaining adjustments to the table (embeds, refreshes, and erases) by other SQL articulations are prohibited.

1.11.3 NOVALIDATE

View Constraints

Prophet doesn't uphold view limitations. Nonetheless, procedure on sees are dependent upon the respectability imperatives characterized on the basic base tables. This implies that you can uphold imperatives on sees through requirements on base tables.

Restrictions on View Constraints

View requirements are a subset of table imperatives and are dependent upon the accompanying limitations:

You can determine just exceptional, essential key, and unfamiliar key requirements on sees. Be that as it may, you can characterize the view utilizing the WITH CHECK OPTION statement, which is comparable to determining a really look at requirement for the view.

Since view imperatives are not implemented straightforwardly, you can't determine INITIALLY DEFERRED or DEFERRABLE.

View imperatives are upheld just in DISABLE NOVALIDATE mode. You should indicate the watchwords DISABLE NOVALIDATE when you pronounce the view requirement, and you can't determine some other mode.

You can't determine the using_index_clause, the exceptions_clause statement, or the ON DELETE condition of the references_clause.

You can't characterize view requirements on traits of an item section.

1.11.4 Using Indexes to Enforce Constraints

1. Utilizing Indexes to Enforce Constraints: In Oracle, "requirements" are an office to authorize decides to ensure that main permissible information esteems are put away in the data set. A requirement, as the name recommends, puts limitations/keeps an eye on the sort or worth of information that can be put away in the data set table. Prophet gives underneath limitations to uphold information integrity:

   NOT NULL: If, according to business rationale, a section or a bunch of segments in a table cannot permit NULL qualities, then, at that point, NOT NULL limitation can be utilized to forestall this.

2. UNIQUE: If, according to business rationale, a segment or a bunch of sections in a table need to store one of a kind qualities, then, at that point, UNIQUE imperative can be utilized to authorize this standard.

3. PRIMARY KEY: Primary Key imperative is a blend of NOT NULL and UNIQUE requirements. The segment or the arrangement of segments on which Primary Key is characterized will permit just interesting and not invalid qualities.

Check Your Progress

7. List the two categories of operators.
8. What do you understand by Oracle functions?
9. List a few aggregate functions.
10. State about the synonyms.
11. What is data integrity?

1.12 ANSWERS TO 'CHECK YOUR PROGRESS'

1. The three major models of data management are as follows:
   - Hierarchical model
   - Network model
   - Relational model

NOTES

2. The benefits of database management systems are as follows:
   - Data independence
   - Controlled data redundancy
   - Efficient data access
3. Structured Query Language (SQL) is a standard query used for relational database management systems.
4. SQL can be classified into the following forms:
   - Data Definition Language (DLL)
   - Data Manipulation Language (DML)
   - Data Control Language (DCL)
   - Transaction Control Language (TCL)
5. Constraints are restrictions that can be put on data to ensure data integrity.
6. The various constraints provided by Oracle are as follows:
   - Primary key
   - Foreign/Reference key
   - Not null
   - Unique
   - Check
   - Default
7. The two categories of operators are as follows:
   - Unary operators
   - Binary operators
8. Oracle functions are built-in functions used for different purposes, such as calculation, comparison and conversion of data.
9. The following listed are aggregate functions that work on a group of values and return a single value:
   - SUM ()
   - MAX ()
   - MIN ()
   - AVG ()
   - COUNT ()
10. A synonym is an elective name for articles like tables, sees, arrangements, put away methods, and other data set items.
11. Data genuinely should keep up with information uprightness, which is adherence to business rules controlled by the data set director or application designer. When planning an information base application, engineers have a few choices for ensuring the respectability of information put away in the data set.

1.13 SUMMARY

NOTES

- The major models proposed to manage data are hierarchical model, network model and relational model.
- A database is a collection of interrelated data stored in two-dimensional structures. It helps organizations to keep records of inventory such as employee details, accounts payable and receivable, and other data.
- Database management includes creating a new database, adding deleting or updating database, retrieval of stored data and putting some constraints on data (e.g., constraint on ID for uniqueness) to maintain data integrity.
- Relational Database Management System (RDBMS) is a collection of database and stored procedures that enable you to store, extract and manage important information from a database. This ensures data security and data integrity in a structured database.
- Tools available for RDBMS include Oracle, INGRES, Sybase, Microsoft SQL Server, MS-Access, IBM-DB-II and My SQL.
- The law of physical data independence states that application programs must be unaffected physical access methods or storage structures are altered.
- The law of logical data independence states that any change to the logical level of table must not affect the application programs in accessing the data.
- The advantages of using RDBMS are data independency, controlled data redundancy, efficient data access, data integrity, data security, data sharing, data consistency, and improved backup and recovery.
- The concept of DBMS is applied in sectors such as banking, aviation, universities, manufacturing, human resources and graphical information systems.
- Oracle database is based on relational data model and Structure Query Language (SQL), a non-procedural language. Oracle database is a tool that supports storing, managing and organization of data.
- Oracle supports two types of SQL: Interactive SQL and PL/SQL.
- The constraints help in maintaining data integrity which is one of the rules defined by E.F. Codd. Constraints could be specified when a table is created or even after the table is created with the ALTER TABLE command.
- There are two types of Oracle functions: aggregate functions and scalar functions.
- Aggregate functions, such as SUM(), MAX(), MIN(), AVG() and COUNT(), work on a group of values (column values) and return a single value.
- A synonym is an elective name for articles like tables, sees, arrangements, put away methods, and other data set items.
- Data genuinely should keep up with information uprightness, which is adherence to business rules controlled by the data set director or application

designer. When planning an information base application, engineers have a few choices for ensuring the respectability of information put away in the data set.

NOTES

1.14 KEY TERMS

- **Constraints:** These are the restrictions that can be placed on data to maintain data integrity.
- **Database:** It is a collection of interrelated data stored in two-dimensional structures that helps organizations to keep records of inventory such as employee details, accounts payable and receivable, and other data.
- **Operators:** They are special characters that manipulate data items (known as operands) to produce results.
- **Oracle database:** It is a tool that supports storing, managing and organization of data.
- **Relational Database Management System (RDBMS):** It is a collection of database and stored procedures that enable the storage, extraction and management of important information from a database. This ensures data security and data integrity in a structured database.
- **Structured Query Language (SQL):** It is a standard query language used for management of all relational database management systems.
- **SQL*Plus:** An Oracle product, it is a command line tool that allows a user to execute SQL statements against an Oracle database.
- **VALIDATE:** The conduct of VALIDATE and NOVALIDATE consistently relies upon whether the requirement is empowered or impaired, either unequivocally or as a matter of course.

1.15 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Differentiate between hierarchical model and network model. How is data represented in these models?
2. List the benefits of using this model over the other models.
3. Write the application of DBMS.
4. How can data integrity be maintained?
5. What are data constraints?
6. Define the term sequence.
7. What do you understand by the term indexes?
8. State the types of integrity constraints.
9. Differentiate between VALIDATE and NOVALIDATE constraints.
10. Mention the indexes using enforce constraints.

NOTES

Long-Answer Questions

1. What are the major models proposed to manage data? Explain advantages and advantages of these models.
2. Explain the twelve rules given by the E. F. Codd for a true relational database management system.
3. What is Oracle? Explain the various modules of Oracle and their functionalities.
4. What is Structured Query Language? Explain the types of SQL commands.
5. Describe the various types of data constraints by giving appropriate examples.
6. Explain the various operators in Oracle with suitable examples.
7. What is Oracle function? Discuss the various types of functions with suitable examples.
8. Explain the synonyms by giving appropriate example.
9. Discuss the indexes with the help of example.
10. Discuss references clause and SCOPE REF constraints by giving appropriate examples.

1.16 FURTHER READING

Snowdon. 1998. *Oracle Programming With Visual Basic.* India: John Wiley & Sons.

Ying Bai. 2021. *Oracle Database Programming with Visual Basic.*NET. India: Wiley-IEEE Press. First Edition.

Byrla. 2017. *Oracle Database 12C*. India: McGraw Hill Education. First Edition.

P.S Deshpande. 2011. *SQL & PL/ SQL for Oracle 11g.* India: Dreamtech Press.

UNIT 2 SQL* PLUS REPORTS, COMMANDS, LOADER AND ACCESSING REMOTE DATABASE

Structure

2.0 INTRODUCTION

SQL*Loader allows you to load data from an external file into a table in the database. It can parse many delimited file formats, such as CSV, tab-delimited, and pipe-delimited. SQL*Loader is a bulk loader utility used for moving data from external files into the Oracle database. Its syntax is similar to that of the DB2 Load utility, but comes with more options. SQL*Loader supports various load formats, selective loading, and multi-table loads.

A database link is a pointer that defines a one-way communication path from an Oracle Database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry. A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique global database name in the network domain. The global database name uniquely identifies a database server in a distributed system.

In this unit you will study about the formatting SQL plus report and commands, clarifying report with spacing and summary lines, introduction SQL*Loader, introduction to database links, counting database links for remote queries and dynamic links, SQL PLUS copy command.

NOTES

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Describe the formatting SQL plus report and commands
- Explain the clarifying report with spacing and summary lines
- Discuss the features and input and output of SQL*Loader
- Explain the concept of database links
- Describe database links for remote queries and dynamic links

2.2 FORMATTING SQL *PLUS REPORT AND COMMANDS

Through the SQL*Plus COLUMN interest, you can change the part headings and reformat the section data in your solicitation results.

Changing Column Headings

While showing district headings, you can either use the default heading or you can change it using the COLUMN interest. The going with regions depict how default headings are determined and how to transform them using the COLUMN interest.

Default Headings

SQL*Plus uses segment or explanation names as default piece headings when showing question results. Area names are every so often short and faint, in any case, and verbalizations can be hard to grasp.

Changing Default Headings

You can portray a more huge region heading with the HEADING state of the COLUMN interest, in the going with plan:

Locale column_name HEADING column_heading

Example 2.1: Changing a Column Heading

To make a report from EMP_DETAILS_VIEW with new headings displayed for LAST_NAME, SALARY, and COMMISSION_PCT, enter the going with orders:

```
Fragment LAST_NAME HEADING 'LAST NAME'
Area SALARY HEADING 'Month to month SALARY'
Area COMMISSION_PCT HEADING COMMISSION
SELECT LAST_NAME, SALARY, COMMISSION_PCT
FROM EMP_DETAILS_VIEW
WHERE JOB_ID='SA_MAN';
LAST NAME MONTHLY SALARY COMMISSION
_____ _____ _____
```

Russell 14000 .4

Assistants 13500 .3

Errazuriz 12000 .3

Cambrault 11000 .3

Zlotkey 10500 .2

To change a segment hustling toward something like two words, encase the new heading in single or twofold clarifications when you enter the COLUMN interest. To show a part heading on more than one line, use a vertical bar (|) where you want to begin a substitute line.

Example 2.2: Splitting a Column Heading

To give the districts SALARY and LAST_NAME the headings MONTHLY SALARY and LAST NAME openly, and to restrict the new headings onto two lines, enter

District SALARY HEADING 'MONTHLY|SALARY'

Section LAST_NAME HEADING 'LAST|NAME'

After a short time rerun the solicitation with the cut (/) request: /

LAST MONTHLY

NAME SALARY COMMISSION

_____ _____ _____

Russell 14000 .4

Partners 13500 .3

Errazuriz 12000 .3

Cambrault 11000 .3

Zlotkey 10500 .2

Example 2.3: Setting the Underline Character

To change the individual used to underline headings to a comparable sign and rerun the solicitation, enter the going with orders:

SET UNDERLINE =

/

LAST MONTHLY

NAME SALARY COMMISSION

========== ========== ==========

Russell 14000 .4

Embellishments 13500 .3

Errazuriz 12000 .3

Cambrault 11000 .3

Zlotkey 10500 .2

In a little while change the underline character back to a scramble:

SET UNDERLINE '- '

Organizing NUMBER Columns

When showing NUMBER parts, you can either see the SQL*Plus default show width or you can change it using the COLUMN interest. Later areas depict the default show and how you can transform it with the COLUMN interest. The association model will remain generally until you enter another, reset the part's plan with

Part column_name CLEAR

Clearly exit from SQL*Plus.

Default Display

A NUMBER part's width moves to the width of the heading or the width of the FORMAT in any case one space for the sign, whichever is more discernible. If you don't explicitly use FORMAT, then, the piece's width will always be on a very basic level the value of SET NUMWIDTH.

SQL*Plus generally shows numbers with at any rate different digits as are required for precision, up to a standard part width constrained by the value of the NUMWIDTH variable of the SET sales (consistently 10). Accepting that a number is more unmistakable than the value of SET NUMWIDTH, SQL*Plus assembles the number or down to the most mind boggling number of characters allowed if possible, or shows hashes expecting the number is unnecessarily colossal.

You can pick a substitute relationship for any NUMBER locale by using an arrangement model in a COLUMN interest. A game-plan model is a depiction of the way wherein you want the numbers in the part to appear, using 9s to address digits.

Changing the Default Display

The COLUMN request sees the segment you want to gather and the model you want to use, as shown:

Segment column_name FORMAT model

Use arrangement models to add commas, dollar signs, point areas (around disgusting traits), and driving zeros to numbers in a given piece. You can other than change the characteristics to a given number of decimal spots, offer short hints to the right of negative credits (rather than to the left), and show regards in stunning documentation.

Example 2.4: Formatting a NUMBER Column

To offer SALARY with a dollar hint, a comma, and the numeral zero rather than a reasonable for any zero characteristics, enter the going with request:

Locale SALARY FORMAT $99,990

Now rerun the current requesting:

/

LAST MONTHLY

NAME SALARY COMMISSION

Russell $14,000 .4

Enhancements $13,500 .3

Errazuriz $12,000 .3

Cambrault $11,000 .3

Zlotkey $10,500 .2

Use a zero in your game plan model, as shown, when you use various courses of action, for instance, a dollar sign and wish to show a zero rather than a certain for zero characteristics.

Organizing Datatypes

While showing datatypes, you can either see the SQL*Plus default show width or you can change it using the COLUMN interest. The plan model will stay thusly until you enter another, reset the part's relationship with

Section column_name CLEAR of course exit from SQL*Plus. Datatypes, in this manual, consolidate the going with sorts:

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

Default Display

The default width of datatype parts is the width of the piece in the data base. The part width of a LONG, CLOB, NCLOB or XMLType defaults to the value of SET LONGCHUNKSIZE or SET LONG, whichever is the more straightforward.

The default width and plan of unformatted DATE sections in SQL*Plus is constrained by the educational arrangement NLS_DATE_FORMAT limit. In any case, the default plan width is A9. See the FORMAT strategy of the part interest for additional information on fixing DATE fragments.

Left side interest is the default for datatypes.

Changing the Default Display

You can change the showed width of a datatype or DATE, by using the COLUMN interest with a system model containing the letter A (for alphanumeric) followed by a number looking out for the width of the piece in characters.

Inside the COLUMN interest, see the part you want to arrange and the model you really want to use:

Fragment column_name FORMAT model

If you show a width more restricted than the segment heading, SQL*Plus abbreviates the heading.

NOTES

Example 2.5: Formatting a Character Column

To set the width of the piece LAST_NAME to four characters and rerun the current solicitation, enter

```
Segment LAST_NAME FORMAT A4
/
LAST MONTHLY
NAME SALARY COMMISSION
____ _________ ___________

Russ $14,000 .4
ell
Part $13,500 .3
ners


Erra $12,000 .3
zuri
z
LAST MONTHLY
NAME SALARY COMMISSION
____ _________ ___________

Camb $11,000 .3
raul
t
Zlot $10,500 .2
key
```

If the WRAP variable of the SET requesting is set to ON (its default regard), the specialist names wrap to the going with line after the fourth individual, as shown in model 5. In the occasion that WRAP is set to OFF, the names are shortened (wipe out) later the fourth individual.

The plan variable WRAP controls everything parts; you can override the setting of WRAP for a given piece through the WRAPPED, WORD_WRAPPED, and TRUNCATED blueprints of the COLUMN interest.

NCLOB or multibyte CLOB parts can't be set up with the WORD_WRAPPED decision. Expecting you plan a NCLOB or multibyte CLOB fragment with COLUMN WORD_WRAPPED, the piece data behaves like COLUMN WRAPPED was applied considering everything.

Now return the part to its past course of action:

Fragment LAST_NAME FORMAT A10

Example 2.6: Formatting a XMLType Column

Going before appearance how to orchestrate an XMLType locale, you should make a table with an XMLType portion definition, and advancement a few data into the table. You can make an XMLType piece like some other customer depicted area. To make a table containing an XMLType district, enter

```
Make TABLE stockrooms (
 warehouse_id NUMBER(3),
 warehouse_spec SYS.XMLTYPE,
 warehouse_name VARCHAR2 (35),
 location_id NUMBER(4));
```

To present one more record containing warehouse_id and warehouse_spec values into the new stockrooms table, enter

```
Implant into stockrooms (warehouse_id, warehouse_spec)
 VALUES (100, sys.XMLTYPE.createXML (
 '<Warehouse whNo="100">
  <Building>Owned</Building>
 </Warehouse>'));
```

To set the XMLType region width to 20 characters and a short period of time later select the XMLType segment, enter

```
District Building FORMAT A20
SELECT
 w.warehouse_spec.extract('/Warehouse/Building/text()').getStringVal()
 "Building"
 FROM stockrooms w;
Building
```

Had

Copying Column Display Attributes

Exactly when you really need to give more than one segment a comparative show credits, you can reduce the length of the orders you ought to enter by using the LIKE state of the COLUMN interest. The LIKE arrangement desires SQL*Plus to copy the show credits of a previously portrayed region to the new part, beside changes made by various verbalizations in a relative requesting.

Example 2.7: Copying a Column's Display Attributes

To give the piece COMMISSION_PCT a close to flaunt credits you obliged SALARY, yet to pick a substitute heading, enter the going with request:

Portion COMMISSION_PCT LIKE SALARY HEADING BONUS

NOTES

Rerun the solicitation:

/

LAST MONTHLY

NAME SALARY BONUS

_____ _____ _____

Russell $14,000 $0

Ruffle $13,500 $0

Errazuriz $12,000 $0

Cambrault $11,000 $0

Zlotkey $10,500 $0

Posting and Resetting Column Display Attributes

To list the current show credits for a given piece, use the COLUMN request followed by the section name just, as shown

Section column_name

To list the current component credits for all parts, enter the COLUMN interest with no piece names or particulars after it

Fragment

To reset the hotshot credits for a part to their default regards, use the CLEAR state of the COLUMN interest as shown

Portion column_name CLEAR

Example 2.8: Resetting Column Display Attributes to their Defaults

To reset all part show credits to their default regards, enter:

CLEAR COLUMNS

Regions cleared

Covering and Restoring Column Display Attributes

You can cover and restore the hotshot credits you have given a specific area. To cover a piece's show credits, enter a COLUMN interest in the going with structure:

Segment column_name OFF

OFF desires SQL*Plus to use the default show credits for the piece, yet doesn't dispose of the characteristics you have portrayed through the COLUMN interest. To restore the characteristics you depicted through COLUMN, use the ON assertion:

Segment column_name ON

SET RECSEP WRAPPED

SET RECSEPCHAR "- "

Finally, enter the going with question:

SELECT LAST_NAME, JOB_TITLE, CITY

NOTES

FROM EMP_DETAILS_VIEW

WHERE SALARY>12000;

After a short time limit the width of the fragment JOB_TITLE and inclination SQL*Plus to wrap whole words to additional lines when major:

Area JOB_TITLE FORMAT A20 WORD_WRAPPED

Run the requesting:

/

LAST_NAME JOB_TITLE CITY

Ruler President Seattle

Kochhar Administration Vice Seattle
President

De Haan Administration Vice Seattle
President

Russell Sales Manager Oxford

Embellishments Sales Manager Oxford

Hartstein Marketing Manager Toronto

6 regions picked.

Enduring you set RECSEP to EACH, SQL*Plus prints a line of characters later every segment (later every office, for the above model).

Going before strategy, set RECSEP to OFF to cover the printing of record separators:

SET RECSEP OFF

2.2.1 Clarifying Your Report with Spacing and Summary Lines

Right when you use an ORDER BY particular in your SQL SELECT sales, lines with a close to worth in the organized piece (or verbalization) are shown together in your result. You can make this outcome more obliging to the customer by using the SQL*Plus BREAK and COMPUTE requesting to make subsets of records and add space or layout lines later every subset.

The piece you pick in a BREAK request is known as a break locale. By evaluating the break district for your ORDER BY blueprint, you make essential subsets of records in your result. You would then have the decision to add intending to the subsets inside an overall BREAK sales, and add a framework line (containing aggregates, midpoints, and so on) by showing the break part in a COMPUTE request.

NOTES

```
SELECT DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY > 12000
Demand BY DEPARTMENT_ID;
DEPARTMENT_ID LAST_NAME SALARY
_____ _____ _____
    20 Hartstein 13000
    80 Russell 14000
    80 Partners 13500
    90 King 24000
    90 Kochhar 17000
    90 De Haan 17000
6 bits picked.
```

To make this report truly obliging, you would use BREAK to encourage DEPARTMENT_ID as the break fragment. Through BREAK you could cover duplicate regards in DEPARTMENT_ID and spot clear lines or start one more page between workplaces. You could consolidate BREAK related with COMPUTE to work out and print outline lines containing the total pay for each office and for all divisions. You could in like manner print outline lines containing the normal, generally over the top, least, standard deviation, unsteadiness, or region count.

Covering Duplicate Values in Break Columns

The BREAK request covers duplicate sees as is consistently done in the piece or explanation you name. Likewise, to cover the duplicate regards in not really settled in an ORDER BY articulation, use the BREAK interest in its most un-complex turn of events:

```
BREAK ON break_column
```

Note:

Whenever you display a segment or explanation in a BREAK interest, use an ORDER BY declaration picking a commensurate piece or enunciation. Expecting you don't do this, breaks happen each time the segment regard changes.

Example 2.9: Suppressing Duplicate Values in a Break Column

To cover the introduction of duplicate office numbers in the solicitation results shown, enter the going with orders:

```
BREAK ON DEPARTMENT_ID;
```

For the going with request (which is the current solicitation set aside in the pad):

```
SELECT DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY > 12000
```

Demand BY DEPARTMENT_ID;

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

20 Hartstein 13000

80 Russell 14000

Accomplices 13500

90 King 24000

Kochhar 17000

De Haan 17000

6 lines picked.

Implanting Space when a Break Column's Value Changes

You can implant clear lines or start another page each time the value changes in the break locale. To implant n clear lines, use the BREAK interest in the going with structure:

BREAK ON break_column SKIP n

To skirt a page, use the requesting in this arrangement:

BREAK ON break_column SKIP PAGE

Example 2.10: Inserting Space when a Break Column's Value Changes

To put one clear line between divisions, enter the going with request:

BREAK ON DEPARTMENT_ID SKIP 1

In the end rerun the solicitation:

/

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

20 Hartstein 13000

80 Russell 14000

Accomplices 13500

90 King 24000

Kochhar 17000

De Haan 17000

6 lines picked.

Presenting Space later Every Row

You may wish to present clear lines or a reasonable page later every line. To skip n lines later every line, use BREAK in the going with structure:

BREAK ON ROW SKIP n

To stay away from a page later every line, use

BREAK ON ROW SKIP PAGE

NOTES

NOTES

Using Multiple Spacing Techniques

Recognize you have more than one piece in your ORDER BY course of action and wish to insert space when each section's worth changes. Each BREAK request you enter replaces the beyond one. Hence, to recall explicit restricting methods for a solitary report or augmentation space later the value changes in more than one facilitated piece, you ought to finish up various locales and exercises in a single BREAK interest.

Example 2.11: Combining Spacing Techniques

Type the going with:

```
SELECT DEPARTMENT_ID, JOB_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000
Demand BY DEPARTMENT_ID, JOB_ID;
```

Now, to skirt a page when the value of DEPARTMENT_ID changes and one line when the value of JOB_ID changes, enter the going with request:

```
BREAK ON DEPARTMENT_ID SKIP PAGE ON JOB_ID SKIP 1
```

To show that SKIP PAGE has made outcomes, make a TTITLE with a page number:

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

Run the new solicitation to see the results:

```
                    Page: 1
DEPARTMENT_ID JOB_ID LAST_NAME SALARY
_____________ _____ _____ _____

        20 MK_MAN Hartstein 13000

                    Page: 2
DEPARTMENT_ID JOB_ID LAST_NAME SALARY
_____________ _____ _____ _____

        80 SA_MAN Russell 14000
                Associates 13500
                    Page: 3
DEPARTMENT_ID JOB_ID LAST_NAME SALARY
_____________ _____ _____ _____

        90 AD_PRES King 24000


          AD_VP Kochhar 17000
                De Haan 17000
6 lines picked.
```

Posting and Removing Break Definitions

NOTES

Going before strategy, switch off the top title show without changing its definition:

TTITLE OFF

You can list your current break definition by entering the BREAK interest with near no details:

BREAK

You can take out the current break definition by entering the CLEAR sales with the BREAKS plan:

CLEAR BREAKS

You may wish to put the requesting CLEAR BREAKS at the beginning of each content to ensure that really entered BREAK orders won't influence questions you run in a given record.

Dealing with Summary Lines when a Break Column's Value Changes

If you organize the spaces of a report into subsets with the BREAK interest, you can perform various evaluations on the lines in each subset. You do this with the pieces of the SQL*Plus COMPUTE request. Use the BREAK and COMPUTE orders together in the going with structures:

BREAK ON break_column

Register work LABEL label_name OF locale section

... ON break_column

You can join specific break packages and exercises, for instance, skipping lines in the BREAK interest, as long as the part you name after ON in the COMPUTE request also appears later ON in the BREAK interest. To review diverse break districts and exercises for BREAK while joining it related with COMPUTE, use these orders in the going with structures:

BREAK ON break_column_1 SKIP PAGE ON break_column_2 SKIP 1

Register work LABEL label_name OF district section

... ON break_column_2

The COMPUTE request has no effect without a seeing BREAK interest.

You can COMPUTE on NUMBER parts and, in express cases, on a wide degree of segments.

Example 2.12: Computing and Printing Subtotals

To deal with the completely out of SALARY by office, first once-over the current BREAK definition:

BREAK

which shows current BREAK definitions:

break on DEPARTMENT_ID page nodup

on JOB_ID keep away from 1 nodup

Now enter the going with COMPUTE sales and run the current sales:

Figure SUM OF SALARY ON DEPARTMENT_ID

/

NOTES

DEPARTMENT_ID JOB_ID LAST_NAME SALARY

20 MK_MAN Hartstein 13000

********** ********** **********

complete 13000

DEPARTMENT_ID JOB_ID LAST_NAME SALARY

80 SA_MAN Russell 14000

Partners 13500

********** ********** **********

complete 27500

DEPARTMENT_ID JOB_ID LAST_NAME SALARY

90 AD_PRES King 24000

AD_VP Kochhar 17000

De Haan 17000

********** ********** **********

complete 58000

6 regions picked.

To enlist how much remunerations for divisions 10 and 20 without printing the cycle mark:

District DUMMY NOPRINT;

Register SUM OF SALARY ON DUMMY;

BREAK ON DUMMY SKIP 1;

SELECT DEPARTMENT_ID DUMMY,DEPARTMENT_ID, LAST_NAME, SALARY

FROM EMP_DETAILS_VIEW

WHERE SALARY>12000

Demand BY DEPARTMENT_ID;

DEPARTMENT_ID LAST_NAME SALARY

20 Hartstein 13000

13000

80 Russell 14000

80 Partners 13500

27500

90 King 24000

90 Kochhar 17000

90 De Haan 17000

58000

6 lines picked.

To work out the remuneration unequivocally close to the completion of the report:

Fragment DUMMY NOPRINT;

Process SUM OF SALARY ON DUMMY;

BREAK ON DUMMY;

SELECT NULL DUMMY,DEPARTMENT_ID, LAST_NAME, SALARY

FROM EMP_DETAILS_VIEW

WHERE SALARY>12000

Demand BY DEPARTMENT_ID;

DEPARTMENT_ID LAST_NAME SALARY

---

20 Hartstein 13000

80 Russell 14000

80 Partners 13500

90 King 24000

90 Kochhar 17000

90 De Haan 17000

98500

6 fragments picked.

Definitively when you encourage the course of action of a NUMBER part, you ought to consider the size of the totals related with the report.

Selecting Summary Lines close to the End of the Report

You can work out and print dynamic lines subject to all characteristics in a piece by recollecting BREAK and COMPUTE for the going with structures:

BREAK ON REPORT

Figure work LABEL label_name OF piece region

... ON REPORT

Example 2.13: Computing and Printing a Grand Total

NOTES

To learn and print the astonishing outright of pay rates for all sales reps and change the cycle name, first enter the going with BREAK and COMPUTE orders:

BREAK ON REPORT

Figure SUM LABEL TOTAL OF SALARY ON REPORT

Then, at that point, enter and run another solicitation:

SELECT LAST_NAME, SALARY

FROM EMP_DETAILS_VIEW

WHERE JOB_ID='SA_MAN';

LAST_NAME SALARY

_____ _____

Russell 14000

Accomplices 13500

Errazuriz 12000

Cambrault 11000

Zlotkey 10500

_____

Without a doubt 61000

To print a stunning total (or amazing OK, remarkable for the most part critical, and so forth) in any case subtotals (or sub-midpoints, and so on), join a break segment and an ON REPORT game-plan in your BREAK interest. Then, enter one COMPUTE interest for the break district and one more to figure ON REPORT:

BREAK ON break_column ON REPORT

Register work LABEL label_name OF section ON break_column

Register work LABEL label_name OF district ON REPORT

Selecting Multiple Summary Values and Lines

You can figure and print for all intents and purposes indistinguishable kind of once-over regard on different parts. To do in that restrict, enter another COMPUTE interest for each part.

Example 2.14: Computing the Same Type of Summary Value on Different Columns

To print the completely out of pay rates and commissions for all specialists, first enter the going with COMPUTE request:

Register SUM OF SALARY COMMISSION_PCT ON REPORT

You don't have to enter a BREAK interest; the BREAK you entered in Example-2.13, "Signing up and Printing a Grand Total" is still for the most part.

Now, change the fundamental line of the select solicitation to join COMMISSION_PCT:

1

1* SELECT LAST_NAME, SALARY

Secure , COMMISSION_PCT;

Finally, run the redressed solicitation to see the results:

/

LAST_NAME SALARY COMMISSION_PCT

_____ _____ _____

Russell 14000 .4

Accomplices 13500 .3

Errazuriz 12000 .3

Cambrault 11000 .3

Zlotkey 10500 .2

_____ _____

full scale 61000 1.5

You can also print unmistakable reasonable lines on a comparative break area. To do thusly, survey the limit concerning each layout line for the COMPUTE request as follows:

Process work LABEL label_name work

Name label_name work LABEL label_name ...

OF locale ON break_column

Enduring you merge various locales later OF and before ON, COMPUTE works out and prints regards for each part you pick.

Example 2.15: Computing Multiple Summary Lines on the Same Break Column

To learn the generally OK and extent of pay for the work pack, first enter the going with BREAK and COMPUTE orders:

BREAK ON DEPARTMENT_ID

Register AVG SUM OF SALARY ON DEPARTMENT_ID

Now, enter and run the going with request:

SELECT DEPARTMENT_ID, LAST_NAME, SALARY

FROM EMP_DETAILS_VIEW

WHERE DEPARTMENT_ID = 30

Demand BY DEPARTMENT_ID, SALARY;

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

30 Colmenares 2500

NOTES

Himuro 2600

Tobias 2800

Baida 2900

Khoo 3100

Raphaely 11000

********** _____

avg 4150

full scale 24900

6 portions picked.

Posting and Removing COMPUTE Definitions

You can list your present COMPUTE definitions by entering the COMPUTE request with near no strategies:

Register

Example 2.16: Removing COMPUTE Definitions

To kill all COMPUTE definitions and the going with BREAK definition, enter the going with orders:

CLEAR BREAKS

breaks cleared

CLEAR COMPUTES

processes cleared

You may wish to put the orders CLEAR BREAKS and CLEAR COMPUTES around the beginning of each content to ensure that really entered BREAK and COMPUTE orders won't influence questions you run in a given record.

2.2.2 Portraying Page and Report Titles and Dimensions

The word page proposes a screen pouring out done with information on your hotshot or a page of a spooled (printed) report. You can put top and base titles on each page, set how much lines per page, and pick the width of each line.

The word report implies the immovable delayed results of a solicitation. You can furthermore put headers and footers on each report and plan them also as top and base titles on pages.

Setting the Top and Bottom Titles and Headers and Footers

As you have now seen, you can set a title to show at the main spot of each page of a report. You can in like manner set a title to show at the lower part of each page. The TTITLE request portrays the top title; the BTITLE request depicts the base title.

You can similarly set a header and footer for each report. The REPHEADER request portrays the report header; the REPFOOTER request portrays the report footer.

A TTITLE, BTITLE, REPHEADER or REPFOOTER request contains the sales name followed by something like one conditions finishing up a position or plan and a CHAR regard you wish to set in that position or give that arrangement. You can join various procedures of details and CHAR regards:

TTITLE position_clause(s) char_value position_clause(s) char_value ...

BTITLE position_clause(s) char_value position_clause(s) char_value ...

REPHEADER position_clause(s) char_value position_clause(s) char_value ...

REPFOOTER position_clause(s) char_value position_clause(s) char_value ...

For portrayals of all TTITLE, BTITLE, REPHEADER and REPFOOTER conditions, see the TTITLE request and the REPHEADER request.

Example 2.17: Placing a Top and Bottom Title on a Page

To put titles at the top and lower part of each page of a report, enter

TTITLE CENTER -

"Most significant point SALES DEPARTMENT PERSONNEL REPORT"

BTITLE CENTER "Association CONFIDENTIAL"

Now run the current solicitation:

/

Most essential point SALES DEPARTMENT PERSONNEL REPORT

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

30 Colmenares 2500

30 Himuro 2600

30 Tobias 2800

30 Baida 2900

30 Khoo 3100

30 Raphaely 11000

Connection CONFIDENTIAL

6 regions picked.

Example 2.18: Placing a Header on a Report

To put a report header on a substitute page, and to concentrate it, enter

REPHEADER PAGE CENTER 'Awesome WIDGETS'

At last run the current requesting:

/

Which shows the going with two pages of result, with the new REPHEADER displayed on the essential page:

Apex SALES DEPARTMENT PERSONNEL REPORT

Wonderful WIDGETS

NOTES

Connection CONFIDENTIAL

Peak SALES DEPARTMENT PERSONNEL REPORT

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

30 Colmenares 2500

30 Himuro 2600

30 Tobias 2800

30 Baida 2900

30 Khoo 3100

30 Raphaely 11000

Connection CONFIDENTIAL

6 segments picked.

To cover the report header without changing its definition, enter

REPHEADER OFF

Coordinating Title Elements

The report in the past exercises might look genuinely captivating enduring you give the association name more prominent enhancement and spot the sort of report and the division name on one or the other side of a substitute line. It may other than help with decreasing the line size and in this manner center the titles considerably more eagerly around the data.

You can accomplish these improvements by adding a couple of details to the TTITLE request and by resetting the system variable LINESIZE, as the going with model shows.

You would setup have the option to report headers and footers in like way as BTITLE and TTITLE using the REPHEADER and REPFOOTER orders.

Example 2.19: Positioning Title Elements

To redisplay the staff report with a repositioned top title, enter the going with orders:

TTITLE CENTER 'A C M E W I D G E T' SKIP 1 -

Center ========== SKIP 1 LEFT 'Staff REPORT' -

RIGHT 'Effort pack' SKIP 2

SET LINESIZE 60

/

A C M E W I D G E T

==========

Staff REPORT SALES DEPARTMENT

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

30 Colmenares 2500

30 Himuro 2600

30 Tobias 2800

30 Baida 2900

30 Khoo 3100

30 Raphaely 11000

Association CONFIDENTIAL

6 lines picked.

The LEFT, RIGHT, and CENTER determinations place the going with ascribes around the beginning, end, and mark of union of the line. The SKIP strategy inclinations SQL*Plus to drop down something like one lines.

Note that there could be currently no space between the last piece of the results and the base title. The last line of the base title prints on the last line of the page. How much space between the last line of the report and the base title depends on the overall page size, how much lines required by the top title, and how much lines in a given page. In the above model, the top title fuses three a more unmistakable number of lines than the top title in the past model. You will sort out some technique for setting how much lines per page later in this part.

To constantly print n clear lines before the base title, use the SKIP n order at the beginning of the BTITLE request. For example, to skirt one line before the base title in the model above, you could enter the going with request:

BTITLE SKIP 1 CENTER 'Alliance CONFIDENTIAL'

Indenting a Title Element

You can recollect the COL strategy for TTITLE or BTITLE to indent the title segment a specific number of spaces. For example, COL 1 places the going with credits in the standard individual situation, as is indistinct from LEFT, or a don't indent of anything. COL 15 places the title part in the fifteenth individual position, indenting it 14 spaces.

Example 2.20: Indenting a Title Element

To print the alliance name left-agreed with the report name indented five spaces on the going with line, enter

TTITLE LEFT 'Pinnacle WIDGET' SKIP 1 -

COL 6 'Effort group PERSONNEL REPORT' SKIP 2

Now rerun the current solicitation to see the results:

/

Pinnacle WIDGET

Outreach pack PERSONNEL REPORT

DEPARTMENT_ID LAST_NAME SALARY

——————— ——————————— ———————

30 Colmenares 2500

30 Himuro 2600

NOTES

30 Tobias 2800

30 Baida 2900

30 Khoo 3100

30 Raphaely 11000

Association CONFIDENTIAL

6 lines picked.

Entering Long Titles

If you truly need to enter a title more gigantic than 500 characters in length, you can use the SQL*Plus request DEFINE to put the text of each line of the title in another substitution variable:

Portray LINE1 = 'This is the first line...'

Portray LINE2 = 'This is the second line...'

Portray LINE3 = 'This is the third line...'

Then, reference the elements in your TTITLE or BTITLE request as follows:

TTITLE CENTER LINE1 SKIP 1 CENTER LINE2 SKIP 1 -

Center LINE3

Showing System-Maintained Values in Titles

You can show the current page number and other system stayed aware of attributes in your title by entering a development consider name to be a title part, for example:

TTITLE LEFT structure maintained_value_name

There are five structure stayed aware of attributes you can show in titles, the most generally used of which is SQL.PNO (the current page number). See TTITLE for a fast outline of configuration stayed aware of qualities you can show in titles.

Example 2.21: Displaying the Current Page Number in a Title

To show the current page number at the most raised indication of each page, close by the connection name, enter the going with request:

TTITLE LEFT 'Peak WIDGET' RIGHT 'PAGE:' SQL.PNO SKIP 2

Now rerun the current requesting:

/

ACMEWIDGET PAGE: 1

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

30 Colmenares 2500

30 Himuro 2600

30 Tobias 2800

30 Baida 2900

30 Khoo 3100

30 Raphaely 11000

Connection CONFIDENTIAL

6 lines picked.

Note that SQL.PNO has a course of action ten spaces wide. You can change this relationship with the FORMAT an area of TTITLE (or BTITLE).

NOTES

Example 2.22: Formatting a System-Maintained Value in a Title

To close everything down space between the word PAGE: and the page number, return the TTITLE request as shown:

TTITLE LEFT 'Highest point WIDGET' RIGHT 'PAGE:' FORMAT 999 -

SQL.PNO SKIP 2

Ultimately rerun the requesting:

/

Highest point WIDGET 'PAGE:' 1

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

30 Colmenares 2500

30 Himuro 2600

30 Tobias 2800

30 Baida 2900

30 Khoo 3100

30 Raphaely 11000

Connection CONFIDENTIAL

6 lines picked.

Posting, Suppressing, and Restoring Page Title Definitions

To list a page title definition, enter the fitting title interest with next to no details:

TTITLE

BTITLE

To cover a title definition, enter:

TTITLE OFF

BTITLE OFF

These orders make SQL*Plus quit showing titles on reports, but don't get the current definitions liberated from the titles. You may restore the current definitions by entering:

TTITLE ON

BTITLE ON

Showing Column Values in Titles

NOTES

You may wish to make an educated authority/detail report that shows a changing master fragment regard at the most raised quality of each page with the detail question results for that value under. You can reference a portion regard in a top title by managing the best worth in a variable and proposing the variable in a TTITLE request. Use the going with sort of the COLUMN requesting to portray the variable:

Area column_name NEW_VALUE variable_name

You should audit the master area for an ORDER BY condition and in a BREAK request using the SKIP PAGE verbalization.

Example 2.23: Creating a Master/Detail Report

Recognize you really want to make a report that shows two explicit supervisors' laborer numbers, each at the most raised reason for a substitute page, and people offering all due appreciation to the boss in complete understanding as the executive's representative number. First make a variable, MGRVAR, to hold the value of the current chairman's delegate number:

Piece MANAGER_ID NEW_VALUE MGRVAR NOPRINT

Since you will basically show the managers' master numbers in the title, you shouldn't play with them to print as a piece of the detail. The NOPRINT condition you entered above tells SQL*Plus not to print the segment MANAGER_ID.

Then, at that point, review a name and the motivation for your page title, enter the legitimate BREAK requesting, and cover the base title from the last model:

TTITLE LEFT 'Chief: ' MGRVAR SKIP 2

BREAK ON MANAGER_ID SKIP PAGE

BTITLE OFF

Finally, enter and run the going with request:

SELECT MANAGER_ID, DEPARTMENT_ID, LAST_NAME, SALARY

FROM EMP_DETAILS_VIEW

WHERE MANAGER_ID IN (101, 201)

Demand BY MANAGER_ID, DEPARTMENT_ID;

Boss: 101

DEPARTMENT_ID LAST_NAME SALARY

_____ _____ _____

10 Whalen 4400

40 Mavris 6500

70 Baer 10000

100 Greenberg 12000

110 Higgins 12000

Boss: 201

DEPARTMENT_ID LAST_NAME SALARY

20 Fay 6000

6 parts picked.

To print the value of a piece at the lower part of the page, you can use the COLUMN interest in the going with structure:

Section column_name OLD_VALUE variable_name

SQL*Plus prints the base title as a piece of the system related with breaking to another page—only ensuing to noticing the new motivation for the master part Accordingly, enduring you just suggested the NEW_VALUE of the master area, you would get the power for the going with plan of nuances. OLD_VALUE audits the value of the master region that was really before the page break began.

Showing the Current Date in Titles

You can, clearly, date your reports essentially by making a value in the title. This is remarkable for promotion libbed reports, yet to run a close to report over and over, you may clearly truly have to have the date therefore appear when the report is run. You can do this by making a variable to hold the current date.

You can reference the predefined substitution variable _DATE to show the current date in a title as you would another variable.

The date arrangement model you review for your LOGIN record or in your SELECT verbalization picks the association wherein SQL*Plus shows the date. See your Oracle Database SQL Reference for additional information on date arrangement models. See Modifying Your LOGIN File for additional information about the LOGIN record.

You can correspondingly enter these orders insightfully. See COLUMN for additional information.

Setting Page Dimensions

Reliably, a page of a report contains how much clear line(s) set in the NEWPAGE variable of the SET sales, a top title, segment headings, your requesting results, and a base title. SQL*Plus shows a report that is too long to even think about evening consider evening contemplate evening ponder fitting on one page on a couple of moderate pages, each with its own titles and piece headings. How much data SQL*Plus shows on each page depends on the current page perspectives.

The default page perspectives used by SQL*Plus are shown under:

- number of lines before the top title: 1
- number of lines per page, from the top title to the lower part of the page: 14
- number of characters per line: 80

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

You can change the page length with the plan variable PAGESIZE. For example, you may wish to do as such when you print a report.

NOTES

NOTES

To set how much lines between the beginning of each page and the top title, use the NEWPAGE variable of the SET sales:

SET NEWPAGE number_of_lines

Expecting you set NEWPAGE to nothing, SQL*Plus skirts zero lines and shows and prints a formfeed character to begin another page. On most kinds of PC screens, the formfeed character clears the screen and moves the cursor to the beginning of the vital line. Right when you print a report, the formfeed character makes the printer move to the most basic indication of one more piece of paper, whether or not the overall page length isn't actually that of the paper. Enduring that you set NEWPAGE to NONE, SQL*Plus doesn't print a certain line or formfeed between report pages.

To set how much lines on a page, use the PAGESIZE variable of the SET sales:

SET PAGESIZE number_of_lines

You may wish to lessen the line size to think a title properly over your result, or you may have to foster line size for drawing on wide paper. You can change the line width using the LINESIZE variable of the SET sales:

SET LINESIZE number_of_characters

Example 2.24: Setting Page Dimensions

To set the page size to 66 lines, clear the screen (or advance the printer to one more piece of paper) close to the start of each page, and set the line size to 70, enter the going with orders:

SET PAGESIZE 66

SET NEWPAGE 0

SET LINESIZE 70

Ultimately enter and run the going with requesting to see the results:

TTITLE CENTER 'Apex WIDGET PERSONNEL REPORT' SKIP 1 -

Center '01-JAN-2001' SKIP 2

Now run the going with request:

Portion FIRST_NAME HEADING 'FIRST|NAME';

Portion LAST_NAME HEADING 'LAST|NAME';

Area SALARY HEADING 'MONTHLY|SALARY' FORMAT $99,999;

SELECT DEPARTMENT_ID, FIRST_NAME, LAST_NAME, SALARY

FROM EMP_DETAILS_VIEW

WHERE SALARY>12000;

Top WIDGET PERSONNEL REPORT

01-JAN-2001

FIRST LAST MONTHLY

DEPARTMENT_ID NAME SALARY

---

90 Steven King $24,000

90 Neena Kochhar $17,000

90 Lex De Haan $17,000

80 John Russell $14,000

80 Karen Partners $13,500

20 Michael Hartstein $13,000

6 lines picked.

In a little while reset PAGESIZE, NEWPAGE, and LINESIZE to their default regards:

SET PAGESIZE 14

SET NEWPAGE 1

SET LINESIZE 80

To list the current anticipated increments of these elements, use the SHOW interest:

SHOW PAGESIZE

SHOW NEWPAGE

SHOW LINESIZE

Through the SQL*Plus request SPOOL, you can store your sales achieves a report or print them on your PC's default printer.

Managing and Printing Query Results

Send your solicitation results to a record when you genuinely need to transform them with a word processor before printing or audit them for a letter, email, or other file.

To store the results of a solicitation in a record—and still hotshot them on the screen—enter the SPOOL interest in the going with structure:

SPOOL file_name

Expecting you don't follow the filename with a period and a development, SPOOL adds a default report increase to the memorable filename it in this manner record. The default influences with the functioning plan; on most has it is LST or LIS. The development isn't added when you spool to structure made documents, for instance,/dev/invalid and/dev/stderr. See the stage express Oracle documentation obliged your functioning system for additional information.

SQL*Plus continues to spool information to the record until you turn spooling off, using the going with kind of SPOOL:

SPOOL OFF

Making a Flat File

While moving data between different programming things, it is a piece of the time basic for use a "level" document (a functioning plan record with zero opportunity to get out characters, headings, or extra characters embedded). For example, accepting that you don't have Oracle Net, you truly need to make a level record for use with SQL*Loader while moving data from Oracle9i to Oracle Database 10g.

NOTES

To make a level record with SQL*Plus, you at first ought to enter the going with SET requesting:

SET NEWPAGE 0

SET SPACE 0

SET LINESIZE 80

SET PAGESIZE 0

SET ECHO OFF

SET FEEDBACK OFF

SET VERIFY OFF

SET HEADING OFF

SET MARKUP HTML OFF SPOOL OFF

Directly following entering these orders, you use the SPOOL interest as shown in the past segment to make the level record.

The SET COLSEP requesting may be fundamental for plan the areas. For additional information, see the SET requesting.

Sending Results to a File

To store the consequences of a sales in a record—and still show them on the screen—enter the SPOOL interest in the going with structure:

SPOOL file_name

SQL*Plus stores all information displayed on the screen later you enter the SPOOL interest in the report you show.

Sending Results to a Printer

To print question results, spool them to a record as portrayed in the past piece. Then, rather than using SPOOL OFF, enter the requesting in the going with structure:

SPOOL OUT

SQL*Plus stops spooling and copies the substance of the spooled report to your PC's standard (default) printer. SPOOL OUT doesn't obliterate the spool record clearly following printing.

Example 2:25: Sending Query Results to a Printer

To pass on a last report and spool and print the results, make a substance named EMPRPT containing the going with orders.

In any case, use EDIT to make the substance with your functioning structure word processor.

Change EMPRPT

Then, at that point, enter the going with orders into the report, using your verbalization processor:

SPOOL TEMP

CLEAR COLUMNS

NOTES

```
CLEAR BREAKS
CLEAR COMPUTES
Area DEPARTMENT_ID HEADING DEPARTMENT
Area LAST_NAME HEADING 'LAST NAME'
Piece SALARY HEADING 'Month to month SALARY' FORMAT $99,999
BREAK ON DEPARTMENT_ID SKIP 1 ON REPORT
Figure SUM OF SALARY ON DEPARTMENT_ID
Figure SUM OF SALARY ON REPORT
SET PAGESIZE 24
SET NEWPAGE 0
SET LINESIZE 70
TTITLE CENTER 'A C M E W I D G E T' SKIP 2 -
LEFT 'Agent REPORT' RIGHT 'PAGE:' -
Plan 999 SQL.PNO SKIP 2
BTITLE CENTER 'Connection CONFIDENTIAL'
SELECT DEPARTMENT_ID, LAST_NAME, SALARY
FROM EMP_DETAILS_VIEW
WHERE SALARY>12000
Demand BY DEPARTMENT_ID;
SPOOL OFF
```

To see the outcome on your screen, you can similarly add SET TERMOUT OFF to the beginning of the report and SET TERMOUT ON to the farthest furthest extents of the record. Save and close the account in your substance gadget (you will appropriately return to SQL*Plus). Now, run the substance EMPRPT:

```
@EMPRPT
```

SQL*Plus shows the outcome on your screen (alongside in the event that you set TERMOUT to OFF), and spools it to the record TEMP:

```
                    A C M E W I D G E T
Delegate REPORT PAGE: 1
Division LAST NAME MONTHLY SALARY
_____ _____ _____

    20 Hartstein $13,000
**********          _____
outright $13,000
    80 Russell $14,000
      Accomplices $13,500
**********          _____
```

full scale $27,500

90 King $24,000

Kochhar $17,000

De Haan $17,000

****

full scale $58,000

full scale $98,500

Association CONFIDENTIAL

6 lines picked..

2.3 SQL*LOADER

To move external files into the Oracle Databases, the SQL*Loader utility is used. It is also useful to load data from other systems into the Oracle database. It loads data in bulk. The data from any text file can be loaded and inserted into the Oracle database.

SQL* Loader supports different load formats such as **selective loading** and **multi-table loads**.

Features of the SQL*Loader

The features of the SQL*Loader are as follows:

- This utility allows user to load data in bulk.
- This is driven by very powerful data parsing engine. The data parsing engine makes it possible to load data from any kind of data format.
- This allows user to load data into multiple tables in the same load session.
- This utility is very flexible and allows users to select the data on the basis of data types.
- This allows users to manipulate the data by using SQL function before loading it into Oracle database.

Input and Output of SQL*Loader

Input of SQL*Loader

SQL*Loader uses `Control File.` These controls files control the behaviour of SQL*Loader. The SQL*Loader uses this file as an input. This file contains the information about how the data will be loaded into the Oracle database. Control File consists of the table name, column data types and field delimiters.

If all records in a data file have the same length, the data file will become a fixed record format file. The control file could specify that records are in fixed format by specifying the starting byte and ending byte location of each and every field.

Outputs of the SQL*Loader are as follows:

- Oracle Database
- A Log File
- A Bad File
- A Potentially Discard File

Oracle Database

Our purpose to use SQL*Loader is to load external data in to Oracle database. If SQL*Loader runs successfully, then the data is loaded in Oracle database, which is the output of the SQL*Loader.

A Log File

When SQL*loader executes a log file is generated by the system. The SQL*loader execution information is stored in a log file. This file contains the information such as CPU time, elapsed time and summary. After completing each SQL*Loader job, the log file should be seen to get the information about the job whether it has been completed successfully or not.

A Bad File

The execution of SQL*Loader can create a file known as a bad file or reject file. It is here that the loader keeps those records that were rejected because of formatting errors or because they caused Oracle errors. If you have specified that a bad file is to be created, the following applies:

- The rejection of one or more records leads to the creation of the bad file wherein the rejected records are logged.
- In case of no rejected records, the bad file is not created. When this occurs, you must reinitialize the bad file for the next run.
- A bad file overwrites any existing file with the same name; ensure that you do not overwrite a file you wish to retain.

A Potentially Discard File

During execution, SQL*Loader can create a discard file for records that are not in compliance with any of the loading criteria. The records contained in this file are called discarded records. Discarded records fail to satisfy any of the `WHEN` clauses specified in the control file.These records are different from rejected records. It is not essential for discarded records to possess any bad data. No insert is attempted on a discarded record.

The rules that need to be kept in mind while creating a discard file are as follows:

- You have specified a discard filename and one or more records fail to satisfy all of the `WHEN` clauses specified in the control file. (If the discard file is created, it overwrites any existing file with the same name, so be sure that you do not overwrite any file you wish to retain.)
- A discard file cannot be created unless records are discarded. For a discard file to be created, you need to discard records.

SQL*Loader Options

Various options are provided by the SQL*Loader such as bad, bindsize, control, data, etc. These options could be specified either on the command line or within a parameter file.

SQL*Loader options are given in Table 2.1

Table 2.1 *SQL*Loader Options*

Bad	If any record is rejected from the input file, a bad file is created by the system. The record rejection could be due to any reason such as non-unique key or a column with the NOT NULL constraint is null, etc.
Bindsize	It shows the size of the bind array in bytes.
Columnarrayrows	This parameter specifies the number of rows to allocate for direct path column arrays.
Control	This is the name of the control file we discussed earlier. A control file specifies the format of the data to be loaded whether it is fixed length or not.
Data	This parameter specifies the name of the file from where you want to load into Oracle database.
Discard	This is the file in which rejected records are stored. Records are discarded due to failure of condition specified in the where clause. When records are discard due to the fail of condition specified in the where clause, the rejected records are stored in the discard file. The discard parameter the name of the file which contains the discarded rows.
Discardmax	This parameter specifies the maximum number of discards to be allowed.
Load	This specifies number of logical records to be loaded into Oracle database.
Log	This parameter is used to specify the name of the log file. This file contains the summary of the result.
Readsize	This parameter specifies the size of the buffer when reading data from the input file into Oracle database. The value of readsize should match that of bindsize parameter.
Rows	This parameter is used to specify the number of rows to load before a commit is issued. For direct path loads, rows are the number of rows to read from the data file before saving the data in the data files.
Skip	This parameter specifies the number of logical records that are allowed to skip.
Streamsize	This specifies the size of direct path streams in bytes.
Userid	This parameter specifies the Oracle username and password.

2.4 INTRODUCTION TO DATABASE LINKS

An edifying grouping connection is a pointer that depicts a solitary course correspondence way from an Oracle Database server to another educational rundown server. The connection pointer is really portrayed as a part in a data word reference table. To get to the association, you ought to be associated with the close by data base that contains the data word reference segment.

An educational rundown alliance association is single course as in a client related with neighbourhood instructive rundown a can use a connection set aside in illuminating record A to get to information in far off data base B, yet customers related with edifying variety B can't use an essentially indistinguishable relationship with get to data in enlightening arrangement A. To get to data on instructive combination a, then, they ought to depict an alliance that is managed in the data word reference of edifying rundown B.

A data base connection coalition licenses neighbourhood customers to move to data on a distant enlightening variety. For this relationship with occur, each datum set in the streamed system ought to have a striking commonly instructive record name in the alliance space. The all-around instructive arrangement name particularly sees an educational collection server in a passed on structure.

Informational arrangement affiliations are either private or public. If they are private, then, simply the customer who made the connection moves close; enduring they are public, then, all illuminating collection customers approach.

One head contrast among instructive grouping affiliations is the way that relationship with a far off informational rundown occur. Customers access a far off edifying record through the going with sorts of affiliations:

Sort of Link	Description
Related customer link	Users interface as themselves, which initiates that they ought to have a record on the far away data base with a comparative customer name and mystery word as their record on the close by illuminating combination.
Fixed customer link	Users interface using the customer name and mystery key hinted in the association. For example, expecting Jane uses a legitimate customer interface that assistants with the hq data base with the customer name and mystery word scott/secret enunciation, then, she relates as scott, Jane has all of the separations in hq permitted to scott clearly, and all the default occupations that scott has been yielded in the hq illuminating rundown.
Current customer link	A customer interfaces as an overall customer. A close by customer can pass on as an overall customer concerning a set aside strategy, without managing the overall customer's abnormal key in a connection definition. For example, Jane can get to a system that Scott formed, getting to Scott's record and Scott's sythesis on the hq data base. Current customer joins are a piece of Oracle Advanced Security.

2.4.1 Counting Database Links for Remote Queries

A data base association allows a customer or program to get to edifying rundown things like tables and viewpoints from another instructive record.

NOTES

Right when you make an instructive record association, you can get to the tables or viewpoints from the far off data base using the going with model:

table_name@database_link

Code language: SQL (Structured Query Language) (sql)

For example, you can address data from a table in the distant edifying arrangement like it was in the close by server:

SELECT * FROM remote_table@database_link;

Code language: SQL (Structured Query Language) (sql)

While getting to a remote table or view over the data base association, the Oracle informative rundown is going apparently as an Oracle client.

Including a commensurate word to encourage the accentuation for getting the chance to objects through an instructive record association

To chip away at the supplement, you can make a vague for the far off thing gotten to through the data base connection and use this article like it was a close by article.

This sentence structure supports the best technique for making an indistinct for a remote table:

Make SYNONYM local_table

FOR remote_table@database_link;

Code language: SQL (Structured Query Language) (sql)

What's more this solicitation uses the comparative rather than the remote table name with the edifying combination association:

SELECT * FROM local_table;

Code language: SQL (Structured Query Language) (sql)

Prophet CREATE DATABASE LINK clarification

There are two sorts of data base affiliations: public and private.

Private data base affiliations are obvious to the owners while public informational arrangement affiliations are noticeable to all customers in the edifying combination. Along these lines, public data base affiliations may address some potential security risks.

To make a private instructive record connection, you use the CREATE DATABASE LINK verbalization as follows:

Make DATABASE LINK dblink

Gather as one With remote_user IDENTIFIED BY secret verbalization

Using 'remote_database';

Code language: SQL (Structured Query Language) (sql)

In this language structure:

• In any case, finish up the name of the data base relationship after the CREATE DATABASE LINK watchwords.

• Second, give customer and mystery key of the far off edifying collection later the CONNECT TO and IDENTIFIED BY watchwords.

• Finally, finish up the help name of the far off educational variety. Enduring you show fundamentally the educational record name, Oracle will join the data base space to the reason in participation string to outline a through and through help name.

Reliably, you add a part int the tnsnames.ora record and reference it as the remote_database in the USING plan.

The going with certificate supports the best technique for causing the private illuminating rundown to speak with a customer in a far off data base with a full association string.

```
Make DATABASE LINK dblink
  Unite as one With remote_user IDENTIFIED BY secret verbalization
  Using '(DESCRIPTION=
    (ADDRESS=(PROTOCOL=TCP)(HOST=oracledb.example.com)(PORT=1521))
      (CONNECT_DATA=(SERVICE_NAME=service_name))
    )';
```

Code language: SQL (Structured Query Language) (sql)

To make a public data base connection, basically add the PUBLIC verbalization:

```
Uncover DATABASE LINK dblink
  Speak With remote_user IDENTIFIED BY secret word
  Using 'remote_database';
```

Code language: SQL (Structured Query Language) (sql)

Make an edifying rundown connection model

In this model, we will make a data base collaborate with a distant Oracle Database server coordinated in the server 10.50.100.143 with the port 1521 and affiliation name SALES.

Notwithstanding, add the going with locale to tnsnames.ora record in the close by Oracle Database server. Reliably, the tnsnames.ora is coordinated in the stock/NETWORK/ADMIN/under ORACLE_HOME:

```
Bargains =
(Depiction =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 10.50.100.143)(PORT
= 1521))
   (CONNECT_DATA =
  (SERVER = DEDICATED)
  (SERVICE_NAME = SALES_PRD)
   )
)
```

Code language: SQL (Structured Query Language) (sql)

NOTES

Then, at that point, use the CREATE DATABASE LINK explanation to make another private instructive grouping connection that accessories with the SALES data base through effect's record:

Make DATABASE LINK bargains

Speak With skirt IDENTIFIED BY Abcd1234

Using 'Blueprints';

Code language: SQL (Structured Query Language) (sql)

Then, issue the SELECT attestation to address data from the customers table on the SALES instructive collection:

SELECT * FROM customers@sales;

Code language: SQL (Structured Query Language) (sql)

Here is the outcome:

Starting there forward, install one more part into the customers table:

Install INTO customers@sales(customer_id, name, email)

VALUES(2,'XYZ Inc','contact@xyzinc.com');

Code language: SQL (Structured Query Language) (sql)

Finally, question data from the customers table again:

SELECT * FROM customers@sales

Code language: SQL (Structured Query Language) (sql)

The result set is according to the going with:

Prophet Database Link best practices

Here are some proposed methodology using the data base affiliations:

1. Naming show the name of the data base affiliations should reflect the shot at data, not the instructive record server. For example, rather than naming an illuminating collection connection SALES_PRD, etc as SALES.
2. Remote illuminating collection customers: you should make a customer submitted for a data base connection. In like way, you should not give this customer to another person. Enduring you don't follow this, the data base won't work when someone changes the unusual key of the customer or even annihilate it.
3. Use an assist with communicating locale in the tnsnames.ora rather than the illuminating grouping unequivocal bogus name with the genuine that you copy between thing, test, and development conditions, you don't have to reiterate the informational variety connection.

2.4.2 Dynamic Links: Using SQL PLUS Copy Command

Remembering Bulk Dynamic SQL for PL/SQL

Mass SQL passes entire mixes forward and in opposite, not just individual parts. This strategy further makes execution by keeping how much setting switches between the PL/SQL and SQL engines. You can use a lone confirmation rather than a circle that gives a SQL clarification in each cycle.

Using the going with sales, details, and cursor quality, your applications can support mass SQL announcements, then, execute them reliably at run time:

Mass FETCH assertion

Mass EXECUTE IMMEDIATE verbalization

FORALL clarification

Accumulate INTO determination

RETURNING INTO detail

%BULK_ROWCOUNT cursor quality

The static assortments of these approvals, conditions, and cursor quality are discussed in "Diminishing Loop Overhead for DML Statements and Queries with Bulk SQL". Recommend that part for establishment information.

Using Dynamic SQL with Bulk SQL

Mass confining licenses Oracle to annex a variable in a SQL clarification to a strategy of traits. The blend type can be any PL/SQL arrangement type: record by table, settled table, or varray. The strategy parts ought to have a SQL datatype like CHAR, DATE, or NUMBER. Three confirmations support dynamic mass ties: EXECUTE IMMEDIATE, FETCH, and FORALL.

EXECUTE IMMEDIATE

You can use the BULK COLLECT INTO condition with the EXECUTE IMMEDIATE clarification to store regards from each piece of a sales' result set in a substitute course of action.

You can use the RETURNING BULK COLLECT INTO strategy with the EXECUTE IMMEDIATE verbalization to store the potential consequences of an INSERT, UPDATE, or DELETE clarification in a huge load of groupings.

Get

You can use the BULK COLLECT INTO condition with the FETCH disclosure to store regards from each piece of a cursor in a substitute social occasion.

FORALL

You can put an EXECUTE IMMEDIATE verbalization with the RETURNING BULK COLLECT INTO inside a FORALL request. You can store the unavoidable aftereffects of all the INSERT, UPDATE, or DELETE clarifications in a huge load of strategies.

You can pass subscripted game-plan parts to the EXECUTE IMMEDIATE declaration through the USING course of action. You can't connect the subscripted parts directly into the string question to EXECUTE IMMEDIATE; for example, you can't amass an arrangement of table names and make a FORALL explanation where each feature applies to a substitute table.

Occasions of Dynamic Bulk Binds

These pieces contain occasions of dynamic mass binds. You can tie portray factors in a strong solicitation using the BULK COLLECT INTO explanation. As shown in Example 2.26, you can recall that course of action for a mass FETCH or mass EXECUTE IMMEDIATE verbalization.

NOTES

Example 2.26: Dynamic SQL with BULK COLLECT INTO Clause

Articulate

```
TYPE EmpCurTyp IS REF CURSOR;
TYPE NumList IS TABLE OF NUMBER;
TYPE NameList IS TABLE OF VARCHAR2(25);
emp_cv EmpCurTyp;
empids NumList;
enames NameList;
sals NumList;
Start
OPEN emp_cv FOR 'SELECT employee_id, last_name FROM trained professionals';
Get emp_cv BULK COLLECT INTO empids, enames;
CLOSE emp_cv;
EXECUTE IMMEDIATE 'SELECT remuneration FROM agents'
Mass COLLECT INTO sals;
END;
/
```

Basically INSERT, UPDATE, and DELETE clarifications can have yield tie factors. You mass bind them with the RETURNING BULK COLLECT INTO particular of EXECUTE IMMEDIATE, as shown in Example 2.27.

Example 2.27: Dynamic SQL with RETURNING BULK COLLECT INTO Clause

```
Declare
TYPE NameList IS TABLE OF VARCHAR2(15);
enames NameList;
bonus_amt NUMBER := 50;
sql_stmt VARCHAR(200);
Start
sql_stmt := 'UPDATE laborers SET pay = pay + :1
        RETURNING last_name INTO :2';
EXECUTE IMMEDIATE sql_stmt
Using bonus_amt RETURNING BULK COLLECT INTO enames;
END;
/
```

To tie the data factors in a SQL articulation, you can use the FORALL clarification and USING explanation, as shown in Example 2.28. The SQL request can't be a sales.

Example 2.28: Dynamic SQL Inside FORALL Statement

```
Articulate
  TYPE NumList IS TABLE OF NUMBER;
  TYPE NameList IS TABLE OF VARCHAR2(15);
  empids NumList;
  enames NameList;
Start
  empids := NumList(101,102,103,104,105);
  FORALL I IN 1..5
    EXECUTE IMMEDIATE
      'UPDATE experts SET pay = pay * 1.04 WHERE employee_id = :1
       RETURNING last_name INTO :2'
       Using empids(i) RETURNING BULK COLLECT INTO enames;
END;
/
```

Rules for Using Dynamic SQL with PL/SQL

This part lets you know the best strategy for taking advantage of dynamic SQL and how to avoid some ordinary gets.

Note:

While using dynamic SQL with PL/SQL, have any information on the risks of SQL mix, which is a potential security issue. For additional information on SQL blend and expected issues, see Oracle Database Application Developer's Guide - Fundamentals. You can relatively search for "SQL imbuement" on the Oracle Technology Network at http://www.oracle.com/progression/

Building a Dynamic Query with Dynamic SQL

You use three explanations to manage a dynamic multi-line interest: OPEN-FOR, FETCH, and CLOSE. As an issue of first significance, you OPEN a cursor variable FOR a multi-line question. Then, you FETCH areas from the result set separately. Exactly when all of the lines are managed, you CLOSE the cursor variable. For additional information about cursor factors, see "Using Cursor Variables (REF CURSORs)".

When to Use or Omit the Semicolon with Dynamic SQL

While developing a lone SQL explanation in a string, do bar any semicolon close to the end.

While developing a PL/SQL dull square, join the semicolon around the culmination of each PL/SQL verbalization and close to the satisfaction of the astonishing square. For example:

```
Start
  EXECUTE IMMEDIATE 'Start Dbms_output.put_line(''semicolons'');
END;';
```

NOTES

```
END;
/
```

Further making Performance of Dynamic SQL with Bind Variables

Right when you code INSERT, UPDATE, DELETE, and SELECT announcements clearly in PL/SQL, PL/SQL changes the variables into tie factors subsequently, to offer the articulations work capably with SQL. Definitively when you develop such clarifications in strong SQL, you really need to show the difficult situation factors yourself to get a relative show.

In the going with model, Oracle opens a substitute cursor for every specific worth of emp_id. This can prompt resource chitchat and dull appearance as each approval is parsed and dealt with.

```
Make PROCEDURE fire_employee (emp_id NUMBER) AS
Start
  EXECUTE IMMEDIATE
        'Destroy FROM laborers WHERE employee_id = ' ||
TO_CHAR(emp_id);
END;
/
```

You can besides energize execution by using a difficult situation variable, which grants Oracle to reuse close to cursor for different expected increments of emp_id:

```
Make PROCEDURE fire_employee (emp_id NUMBER) AS
Start
  EXECUTE IMMEDIATE
     'Eradicate FROM delegates WHERE employee_id = :id' USING
emp_id;
END;
/
```

Passing Schema Object Names As Parameters

Acknowledge you really need a framework that perceives the name of any educational record table, then, at that point, drops that table from your preparation. You should make a string with an explanation that combines the article names, then, at that point, use EXECUTE IMMEDIATE to execute the affirmation:

```
Make TABLE employees_temp AS SELECT last_name FROM specialists;
Make PROCEDURE drop_table (table_name IN VARCHAR2) AS
Start
 EXECUTE IMMEDIATE 'DROP TABLE ' || table_name;
END;
/
```

Use interface with gather the string, rather than trying to pass the table name as a tough spot variable through the USING course of action.

Similarly, assuming you really want to call a strategy whose name is dim until runtime, you can pass a breaking point seeing the procedure. For instance, the going with methodology can call another procedure (drop_table) by showing the system name when executed.

Make PROCEDURE run_proc (proc_name IN VARCHAR2, table_name IN VARCHAR2) ASBEGIN

EXECUTE IMMEDIATE 'CALL "' || proc_name || '" (:proc_name)' utilizing table_name;

END;

/

To drop a table with the drop_table framework, you can run the methodology as follows. Note that the system name is progressed.

Make TABLE employees_temp AS SELECT last_name FROM delegates;

Start

run_proc('DROP_TABLE', 'employees_temp');

END;

/

Utilizing Duplicate Placeholders with Dynamic SQL

Placeholders in a phenomenal SQL verbalization are associated with tie questionsin the USING condition by position, not by name. Assuming you conclude a movement of placeholders like :a, :a, :b, :b, you should remember four things for the USING decree. For instance, given the momentous string

sql_stmt := 'Supplement INTO finance VALUES (:x, :x, :y, :x)';

the way that the name X is rehashed isn't fundamental. You can code the relating USING specification with four diverse tie factors:

EXECUTE IMMEDIATE sql_stmt USING a, a, b, a;

In the event that the solid verification keeps an eye on a PL/SQL block, the standards for copy placeholders are wonderful. Each extraordinary placeholder guides for something single in the USING verbalization. Expecting a relative placeholder seems twice, all references to that name diverge from one tie debate in the USING declaration. In Example 2.29, all references to the placeholder x are associated with the guideline tie debate a, and the second surprising placeholder y is associated with the subsequent tie struggle b.

Example 2.29: Using Duplicate Placeholders With Dynamic SQL

Make PROCEDURE calc_stats(w NUMBER, x NUMBER, y NUMBER, z NUMBER) IS

Start

DBMS_OUTPUT.PUT_LINE(w + x + y + z);

NOTES

```
END;
/
Articulate
  a NUMBER := 4;
  b NUMBER := 7;
  plsql_block VARCHAR2(100);
Start
  plsql_block := 'Start calc_stats(:x, :x, :y, :x); END;';
  EXECUTE IMMEDIATE plsql_block USING a, b;
END;
/
```

Utilizing Cursor Attributes with Dynamic SQL

The SQL cursor credits %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT work when you issue an INSERT, UPDATE, DELETE, or single-area SELECT articulation in solid SQL:

```
Start
   EXECUTE IMMEDIATE 'Annihilate FROM workers WHERE employee_id > 1000';
   DBMS_OUTPUT.PUT_LINE('Number of workers annihilated: ' || TO_CHAR(SQL%ROWCOUNT));
END;
/
```

Likewise, when joined to a cursor variable name, the cursor credits return data about the execution of a multi-fragment question:

Example 2.30: Accessing %ROWCOUNT For an Explicit Cursor

```
Articulate
  TYPE cursor_ref IS REF CURSOR;
  c1 cursor_ref;
  TYPE emp_tab IS TABLE OF employees%ROWTYPE;
  rec_tab emp_tab;
  rows_fetched NUMBER;
Start
  OPEN c1 FOR 'SELECT * FROM specialists';
  Bring c1 BULK COLLECT INTO rec_tab;
  rows_fetched := c1%ROWCOUNT;
   DBMS_OUTPUT.PUT_LINE('Number of workers got: ' || TO_CHAR(rows_fetched));
END;
/
```

For extra data about cursor credits, see "Regulating Cursors in PL/SQL".

Passing Nulls to Dynamic SQL

The serious NULL isn't permitted in the USING specification. To work around this cutoff, supersede the watchword NULL with a uninitialized variable:

```
Make TABLE employees_temp AS SELECT * FROM EMPLOYEES;
Report
  a_null CHAR(1); - - set to NULL thusly at run time
Start
    EXECUTE IMMEDIATE 'UPDATE employees_temp SET
commission_pct = :x' USING a_null;
END;
/
```

Utilizing Database Links with Dynamic SQL

PL/SQL subprograms can execute dynamic SQL articulations that utilization educational assortment partners with recommend objects on far away information bases:

```
Make PROCEDURE delete_dept (db_link VARCHAR2, dept_id
INTEGER) IS
Start
  EXECUTE IMMEDIATE 'Erase FROM departments@' || db_link ||
    ' WHERE department_id = :num' USING dept_id;
END;
/
— erase division id 41 in the work environments table on the distant DB
hr_db
CALL delete_dept('hr_db', 41);
```

The objections of distant technique calls (RPCs) can contain dynamic SQL announcements. For instance, acknowledge the going with independent cutoff, which returns how much lines in a table, pesters the hr_db information base in London:

```
Make FUNCTION row_count (tab_name VARCHAR2) RETURN
NUMBER AS
  lines NUMBER;
Start
  EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' || tab_name
INTO lines;
  Bring lines back;
END;
/
```

NOTES

— From a mysterious square, you may call the cutoff from a decent ways, as follows:

```
Articulate
  emp_count INTEGER;
Start
  emp_count := row_count@hr_db('employees');
  DBMS_OUTPUT.PUT_LINE(emp_count);
END;
/
```

Utilizing Invoker Rights with Dynamic SQL

Dynamic SQL grants you to make plan the board strategies that can be bound together in one graph, and can be called from different models and work on the things in those outlines. For instance, this strategy can drop any sort of information base article:

```
Make OR REPLACE PROCEDURE drop_it (kind IN VARCHAR2, name IN VARCHAR2)
  AUTHID CURRENT_USER AS
Start
  EXECUTE IMMEDIATE 'DROP ' || kind || ' ' || name;
END;
/
```

Accept that this system is critical for the HR arranging. Without the AUTHID condition, the technique would dependably drop objects in the HR design, paying little cerebrum to who calls it. Whether or not you pass a completely qualified thing name, this procedure would not have the qualifications to make changes in different plans.

The AUTHID course of action lifts both of these obstructions. It allows the system to run with the differentiations of the client that accumulates it, and makes mismatched references infer objects in that client's framework.

For subtleties, see "Utilizing Invoker's Rights Versus Definer's Rights (AUTHID Clause)".

Utilizing Pragma RESTRICT_REFERENCES with Dynamic SQL

A breaking point called from SQL clarifications should submit to unequivocal guidelines wanted to control unplanned effects. (See "Controlling Side Effects of PL/SQL Subprograms".) To check for infringement of the principles, you can utilize the pragma RESTRICT_REFERENCES. The pragma states that a cutoff doesn't examine or make instructive file tables or gathering factors. (For extra data, see Oracle Database Application Developer's Guide - Fundamentals.)

Tolerating the breaking point body contains a solid INSERT, UPDATE, or DELETE clarification, the cutoff dependably excuses the principles make no information base state (WNDS) and read no educational assortment state (RNDS). PL/SQL can't separate those assistant effects typically, considering the way that exceptional SQL articulations are checked at run time, not at complete time. In an EXECUTE IMMEDIATE verbalization, just the INTO condition can be checked at absolute an ideal opportunity for infringement of RNDS.

Keeping away from Deadlocks with Dynamic SQL

In a few conditions, executing a SQL information definition order accomplishes a stop. For instance, the going with reasoning makes a gridlock since it attempts drop itself. To stay away from stops, never attempt to ALTER or DROP a subprogram or gathering while you are now utilizing it.

Make OR REPLACE PROCEDURE calc_bonus (emp_id NUMBER) AS

Start

EXECUTE IMMEDIATE 'DROP PROCEDURE calc_bonus'; - - gridlock!

END;

/

Thusly around Compatibility of the USING Clause

Precisely when a solid INSERT, UPDATE, or DELETE explanation has a RETURNING plan, yield tie questions can go in the RETURNING INTO condition or the USING specification. In new applications, utilize the RETURNING INTO specification. In old applications, you can keep on utilizing the USING announcement.

Utilizing Dynamic SQL With PL/SQL Records and Collections

You can utilize dynamic SQL with records and mixes. As displayed in Example 2.31, you can bring lines from the outcome set of a dynamic multi-line examination concerning a record:

Example 2.31: Dynamic SQL Fetching into a Record

Broadcast

TYPE EmpCurTyp IS REF CURSOR;

emp_cv EmpCurTyp;

emp_rec employees%ROWTYPE;

sql_stmt VARCHAR2(200);

v_job VARCHAR2(10) := 'ST_CLERK';

Start

sql_stmt := 'SELECT * FROM workers WHERE job_id = :j';

OPEN emp_cv FOR sql_stmt USING v_job;

NOTES

```
Circle
 Bring emp_cv INTO emp_rec;
 Leave WHEN emp_cv%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE('Name: ' || emp_rec.last_name || ' Job Id: ' ||
                    emp_rec.job_id);
 END LOOP;
 CLOSE emp_cv;
END;
/
```

For an occasion of using dynamic SQL with object types, see "Using Dynamic SQL With Objects".

Combine efforts with Remote Database.

Communicating Locally with the SQL Command Line

Associate locally infers running the SQL Command Line (SQL*Plus) and Oracle Database XE on an identical PC. There are two procedures for starting a local relationship with the SQL Command Line:

From the workspace

From a terminal get-together (Linux) or sales window (Windows)

Starting the SQL Command Line from the Desktop

To start the SQL Command Line from the workspace and accessory locally:

Do one of the going with:

On Windows: Click Start, feature Programs (or All Programs), incorporate Oracle Database 11g Express Edition, and sometime later select Run SQL Command Line.

On Linux with Gnome: In the Applications menu, feature Oracle Database 11g Express Edition, and subsequently select Run SQL Command Line.

On Linux with KDE: Click the image for the K Menu, feature Oracle Database 11g Express Edition, and thusly select Run SQL Command Line.

The SQL Command Line request window opens.

At the SQL Command Line brief, enter the going with request:

Assistant username/secret word

For example, to relate as customer HR with the perplexing word PEOPLE, enter the going with request:

Associate HR/PEOPLE

Starting the SQL Command Line from a Terminal Session or Command Window

To start the SQL Command Line from a terminal collecting or requesting window and accessory locally:

If not before long open, open a terminal party (Linux) or a sales window (Windows).

(Linux gave that) the vital environment factors are not at present set for your get-together, set them as depicted in "Setting Environment Variables on the Linux Platform".

Enter the going with request at the functioning plan brief:

sqlplus/nolog

At the SQL Command Line brief, enter the going with request:

Associate username/secret explanation

For example, to relate as customer HR with the baffling explanation PEOPLE, enter the going with request:

Associate HR/PEOPLE

See Also:

"About Local and Remote Connections"

Connecting Remotely with the SQL Command Line

Associate remotely proposes running the SQL Command Line (SQL*Plus) on one PC (the distant PC), and as such beginning a relationship with Oracle Database XE on a substitute PC.

To begin a distant relationship from the SQL Command Line using the Oracle Database XE:

On the distant PC, start a terminal get-together (Linux) or open a requesting window (Windows.)

At whatever point provoked for have limits, sign in to the far off PC.

(Linux gave that) the fundamental environment factors are not at present set for your social gathering, set them as portrayed in "Setting Environment Variables on the Linux Platform".

Enter the going with request at the functioning plan brief:

sqlplus/nolog

Enter a CONNECT request at the SQL Command Line brief, giving a reason in collaboration string.

Interface username/password@[//]host[:port][/service_name]

See "About Remote Connections" for a depiction and occasions of point of correspondence strings.

See Also:

"About Local and Remote Connections"

NOTES

Environment Variables Reference for Linux

This part gives reference information to setting environment factors on Linux for the going with two conditions:

Passing on locally

Passing on from a segment from Oracle Database XE. Table 2.2 records the environment factors that you should set for these conditions. Table 2.3 gives environment variable portrayals and required qualities.

Table 2.2 *Required Linux Environment Variables for Connecting with Oracle Utilities*

Association Type	Required Environment Variables
Close by	ORACLE_SID ORACLE_HOME Way NLS_LANG LD_LIBRARY_PATH
Remote, using Oracle Database XE	ORACLE_HOME Way NLS_LANG LD_LIBRARY_PATH SQLPATH

Table 2.3 *Environment Variable Descriptions and Values for Linux*

Variable Name	Description	Required Value
ORACLE_SID	Prophet Instance ID	XE
ORACLE_HOME	Home library Oracle	For neighborhood association: /usr/lib/prophet/xe/application/prophet/thing/11.2.0/server For far away relationship with Oracle Database XE: /usr/lib/prophet/xe/application/prophet/thing/11.2.0/client
Path	Investigate way for executables. (Should add $ORACLE_HOME/ compartment to the way.)	For Bourne, Korn, or Bash shell: $ORACLE_HOME/bin:$PATH For C shell: $ORACLE_HOME/bin:${PATH}
NLS_LANG	Area (language and locale used by client applications and the data base; character set used by client applications)	(The best language, space, and character set. See Oracle Database Express Edition Installation Guide for Linux x86-64 for nuances.) Defaults to AMERICAN_AMERICA.US7ASCII
LD_LIBRARY_PATH	Channel way for shared libraries. (Should add $ORACLE_HOME/lib to the way.	$ORACLE_HOME/lib:$LD_LIBRARY_PATH
SQLPATH	Search way used by the SQL Command Line (SQL*Plus) for *.sql scripts. Contains a colon-isolated layout of ways. Ought to join the space of the site profile script, glogin.sql.	$ORACLE_HOME/sqlplus/director

Model

Coming up next are the Bash shell orders that set the fundamental environment factors for a close by relationship on a Linux foundation in the United States:

ORACLE_SID=XE;export ORACLE_SID

ORACLE_HOME=/usr/lib/prophet/xe/application/prophet/thing/11.2.0/server;export ORACLE_HOME

PATH=$ORACLE_HOME/bin:$PATH;export PATH

NLS_LANG=AMERICAN_AMERICA.AL32UTF8;export NLS_LANG

LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH;export LD_LIBRARY_PATH

Environment Variable Scripts

Prophet Database XE and Oracle Database XE transport with two shell scripts that you can use to obligingly set environment factors. The substance are coordinated in $ORACLE_HOME/compartment and are named as follows:

oracle_env.sh (for Bourne, Korn, or Bash shell)

oracle_env.csh (for C shell)

Check Your Progress

1. What is break locale?
2. What does SQL*Loader use to control its behaviour?
3. Define the term Oracle database.
4. What are private and public affiliations?

2.5 ANSWERS TO 'CHECK YOUR PROGRESS'

1. The piece you pick in a BREAK request is known as a break locale.
2. SQL*Loader uses Control File, which controls behaviour of SQL *Loader.
3. Our purpose to use SQL*Loader is to load external data in to Oracle database.
4. Private data base affiliations are obvious to the owners while public informational arrangement affiliations are noticeable to all customers in the edifying combination. Along these lines, public data base affiliations may address some potential security risks

2.6 SUMMARY

- Through the SQL*Plus COLUMN interest, you can change the part headings and reformat the section data in your solicitation results.
- To move external files into the Oracle Databases, SQL*Loader utility is used. It is also useful to load data from other systems into the Oracle database.

- SQL*Loader supports various load formats such as selective loading and multi-table loads.
- SQL*Loader use Control File, which controls the behaviour of SQL*Loader. It uses this file as an input.
- SQL*Loader provides various options, which can be specified either on the command line or within a parameter file. These parameters are Bad, Bindsize, Columnarrayrows, Control, Data, Discard, Discardmax, Load, Log, etc.
- An edifying grouping connection is a pointer that depicts a solitary course correspondence way from an Oracle Database server to another educational rundown server. The connection pointer is really portrayed as a part in a data word reference table
- Private data base affiliations are obvious to the owners while public informational arrangement affiliations are noticeable to all customers in the edifying combination. Along these lines, public data base affiliations may address some potential security risks.
- A data base association allows a customer or program to get to edifying rundown things like tables and viewpoints from another instructive record.

2.7 KEY TERMS

- **Break locale:** It is the piece you pick in a BREAK request is known as a break locale.
- **SQL *loader:** This is a utility which is used to move external files into the Oracle database.
- **Data base association:** It allows a customer or program to get to edifying rundown things like tables and viewpoints from another instructive record.

2.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Give the definition of posting and removing break.
2. Write the commands of SQL* PLUS.
3. What is the extension of an exported file?
4. What do you understand by the Oracle binary format?
5. What is a dump file?
6. What is the use of EXP_FULL_DATABASE?
7. Write the proposed methodology using the database affiliations.
8. What is remote database?

Long-Answer Questions

1. Explain the Formatting SQL Plus by giving appropriate example.
2. Discuss the report with spacing and summary lines with appropriate example.
3. What are the various inputs and outputs of the SQL*Loader? Explain.
4. What do you understand by the Control file? Why is this file used by the SQL*Loader utility? Discuss.
5. Describe a log file. When is this file created?
6. Explain the database links by giving appropriate example.
7. Explain the counting database links for remote queries with the help of appropriate example.
8. Discuss the SQL PLUS copy commands by giving appropriate example.

2.9 FURTHER READING

Snowdon. 1998. Oracle Programming With Visual Basic. India: John Wiley & Sons.

Ying Bai. 2021. Oracle Database Programming with Visual Basic.NET. India: Wiley-IEEE Press. First Edition.

Byrla. 2017. Oracle Database 12C. India: McGraw Hill Education. First Edition.

P.S Deshpande. 2011. SQL & PL/ SQL for Oracle 11g. India: Dreamtech Press.

UNIT 3 OVERVIEW OF PL/SQL

NOTES

Structure

3.0 INTRODUCTION

The PL/SQL is also known as an embedded SQL and is a superset of SQL. PL/SQL is an acronym of procedural language/structure query language. It supports procedural features and SQL commands. In PL/ SQL programs SQL statements and procedural statements could be combined for various purposes. All the statements are executed from the top to bottom one after another. PL/ SQL supports procedural statements that include the control statements, loops, exception handling and structure query language, procedure and functions. Once a PL/SQL program is written it can be saved on a disk for further use and to increase the reusability of the code.

A stored procedure (simply, a proc) is a named PL/SQL block which carries out one or more specific task. This is the same as a procedure in other programming languages. A procedure contains a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body contains the declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage. A stored function is similar to a stored procedure, except the fact that a function always returns a value to the caller.

In this unit you will study about the PL/SQL syntax and its data types, Oracle database server, advantage in PL/SQL, stored procedures, stored functions and advantage of stored procedure and function.

NOTES

3.1 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand the PL/SQL syntax and its data types
- Explain the Oracle database server
- Know about the advantage in PL/SQL
- Describe the stored procedures
- Define the stored functions
- Discuss about the various advantage of stored procedure and function

3.2 PL/SQL: FUNCTIONS, FEATURES AND STRUCTURE

The PQL/SQL is a program unit, which can be compiled into the oracle database. Procedural language and structured query is one of the three programming languages embedded in the oracle database. It has many features, some of them are as follows:

1. **Better performance:** Unlike SQL, PL/ SQL sends the entire block of code to the Oracle engine at one run that helps in saving the execution time and reduces network traffic as well. Procedural language and SQL could be combined for better performance and to improve efficiency.
2. **Portability:** A PL/ SQL program is not machine dependent. Once an applications is written it could be run on any operating system and platform. Program written with PL/ SQL could be reused by different users on the same or different platforms.
3. **Increased productivity:** In PL/ SQL programs, SQL statements and procedural statements could be combined for various purposes. All the statements are executed from the top to bottom one after the other.

   A PL/ SQL program can be saved further which increases its reusability of code. Its encapsulation, exception handling, control statements, data hiding and other features increase its productivity.
4. **Error handling:** The PL/ SQL has a feature to handle abnormal situations with the exception handling block during program execution. These abnormal situations may be data type error, zero divide error, etc. Handling the error includes catching an error and taking a specified action depending upon the type of exception.
5. **Object-oriented programming support:** Object-oriented programming supports various features such as encapsulation, inheritance, objects and many more features. PL/ SQL also supports object-oriented features which makes the PL/ SQL program maintainable and reusable and closer to the real world.

6. **Security:** The PL/ SQL–stored procedures are mostly stored on the server and used by the client as per the access given to the users. This feature makes the PL/ SQL program secure. For example, a table could be accessed by the user to retrieve the records from the table but it can be restricted to modify the records.

Some of the important points to remember are as follows:

- Only data manipulation language (DML) and transaction control language (TCL) commands are supported by PL/ SQL.
- The PL/ SQL programs can be written in any text editor, i.e. notepad, WordPad, etc.
- Exception handling and declaration sections are optional.
- The program starts with begin statement and ends with end statement.
- Every statement should be terminated with a semicolon (;) .

3.2.1 PL/SQL Functions and Syntax

A function is termed as PL/SQL block which is a subprogram like a procedure. The main difference between a procedure and a function is that a function always returns a value whereas a procedure may or may not return a value. Thus, a function is a program that performs an action and returns a value. Following is the general syntax for creating a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype;
IS
Declaration_section
BEGIN
Execution_section
Return return_variable;
EXCEPTION
exception section
Return return_variable;
END;
```

The header section describes the `RETURN` type of the function. The return_datatype can be any of the Oracle datatypes such as varchar, number, etc. The execution and exception section both should return a value which is of the datatype defined in the header section. A function can be performed in the following ways:

1. It returns a value hence a variable can be assigned to it.
2. It can be used as a part of a `SELECT` statement.
3. It can be used in PL/SQL output statements.
4. It must return a value.
5. It can return data in `OUT` and IN `OUT` parameters.
6. The `RETURN` statement in a function returns control to the calling program to return the results of the function.

NOTES

Following is the syntax for writing local functions:

```
[CREATE  [OR REPLACE ] ]
FUNCTION function_name [ ( parameter [ , parameter ]... )
] RETURN
datatype
   [ AUTHID { DEFINER  | CURRENT_USER } ]
   [ PARALLEL_ENABLE
    [ { [CLUSTER parameter BY (column_name [, column_name
]... ) ] |
     [ORDER parameter BY (column_name [ , column_name
]... ) ] } ]
     [ ( PARTITION parameter BY
     { [ {RANGE | HASH } (column_name [, column_name]...)]
| ANY }
) ]
   ]
     [DETERMINISTIC]       [  PIPELINED    [  USING
implementation_type ] ]
   [ AGGREGATE  [UPDATE VALUE]  [WITH EXTERNAL CONTEXT]
USING  implementation_type ]  {IS | AS}
   [ PRAGMA AUTONOMOUS_TRANSACTION; ]
   [ local declarations ]
BEGIN
  executable statements
[ EXCEPTION
   exception handlers ]
END [ name ];
```

The CREATE clause permits you to create isolated functions that are stored in an Oracle database. The CREATE FUNCTION can be executed from SQL*Plus or from a program by using native dynamic SQL.

The AUTHID clause decides whether a stored function is executed with the rights of its owner (the default) or current user and whether its unskilled locations to schema objects are determined in the schema of the owner or current user. You can supersede the default actions by specifying CURRENT_USER.

The PARALLEL_ENABLE option states that a stored function can be safely used in the related sessions of parallel DML evaluations. The condition of a main (logon) session is on no account shared with related sessions. Each related session has its own specific state which is initialized with the beginning of session.

The DETERMINISTIC permits the optimizer to avoid superfluous function calls. For example, if a stored function is already called before with the same arguments, then the optimizer can decide on the use of the previous result. The function result must not depend on the condition of session variables or schema objects. Only DETERMINISTIC functions are called from a function-based index that has enabled query-rewrite.

NOTES

The pragma AUTONOMOUS_TRANSACTION is used to instruct the PL/SQL compiler for marking a function as autonomous or independent.

Similar to the procedure, a function has two parts: the spec and the body. The function spec starts with the keyword FUNCTION and ends with the RETURN clause specifying the datatype of the return value. Declaration of parameters is optional. Functions without parameters are written without parentheses. The function body initiates with the keyword IS or AS and ends with the keyword END followed by a function name which is optional. The function body has three parts: a declarative part, an executable part and an optional exception-handling part. The declarative part holds local declarations and are placed between the keywords IS and BEGIN. The DECLARE keyword is not used. The executable part holds statements to be placed between the keywords BEGIN and EXCEPTION or END. In the executable part of a function one or more RETURN statements must be used. The exception-handling part holds exception handlers to be placed between the keywords EXCEPTION and END.

Table Functions

Table functions are the specific functions that create a collection of rows in the form of a nested table or a VARRAY to be queried to a physical database table or allocated to a PL/SQL collection variable. A table function is used as the name of a database table in the FROM clause of a query or as a column name in the SELECT list of a query. A table function uses a collection of rows as input. Table function execution can be parallelized and returned rows can be directly streamed to the next process without intermediate staging. Rows that are returned by a table function can be pipelined. Streaming, pipelining and parallel execution of table functions progress performance.

Figure 3.1 shows a characteristic data-processing situation in which the data passes via three transformations implemented with table functions before these are loaded into a database finally. In this situation, the table functions are not parallelized and the whole result compilation is staged in a temporary table after each transformation.

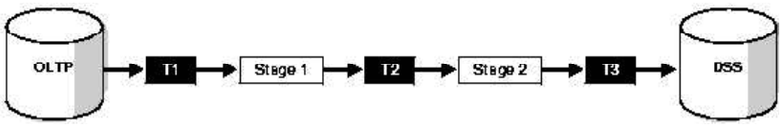


Fig. 3.1 *Typical Data Processing with Unparallelized, Unpipelined Table Functions*

Figure 3.2 shows that the same situation can be streamlined by using streaming and parallel execution.

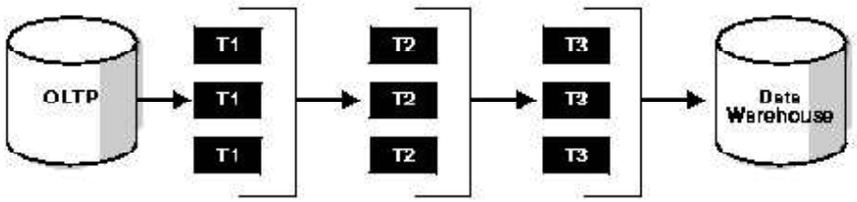


Fig. 3.2 *Data Processing Using Pipelining and Parallel Execution*

NOTES

Data is termed pipelined when it is immediately consumed by a consumer (transformation) as the producer (transformation) produced and is not staged in tables or a cache before it inputs to the next transformation. Pipelining facilitates a table function to return rows more rapidly and can also reduce the memory needed to cache the result of a table function.

3.2.2 Structure of PL/SQL Program

A PL/ SQL program block is divided into three sections (Figure 3.3):

1. Declaration section
2. Execution section
3. Exception handling section

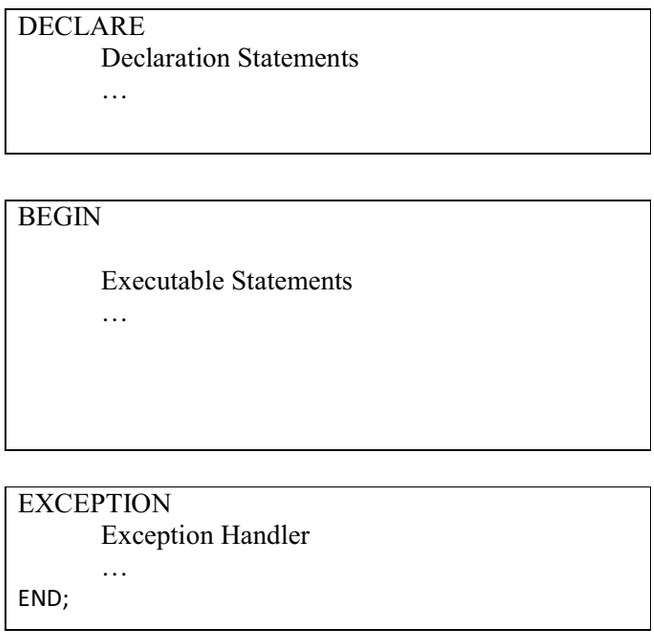


Fig. 3.3 *Structure of PL/SQL Block*

1. **Declaration Section:** In declaration section variables, constants, user defined exceptions, cursor and other objects are declared. This is an optional section. This section begins with the key word `DECLARE`.
2. **Execution section:** All the executable statements such as SQL statements, control statements, loops are written under this section. This is a mandatory section. This section begins with the key word `BEGIN` and ends with the key word `END`.
3. **The exception handling section:** During program execution many abnormal situations may occur. To handle those situations statements are written in this block. These situations are known as errors which occur due to the logical error, syntax error or system error. This is an optional section.

   The PL/ SQL Syntax is as follows :

```
DECLARE
  declaration_statements
```

```
  …
BEGIN
  executable_statements
  …
EXCEPTION
  exception_handling_statements
  …
END ;
```

The PL/ SQL Engine

Oracle uses a PL/ SQL engine to process the PL/ SQL statements. Either the PL/ SQL program is stored on the client side or on the server side. PL/ SQL engine is used by Oracle to execute the program statements.

3.2.3 Oracle Database Server

An Oracle database is defined as a compilation/collection of data defined as a unit. The basic principle of a database system is to store and retrieve related information. A database server is considered as the key to solve the problems related to information management. As a general rule, a server consistently/reliably manages a huge amount of data in a multiuser environment so that various users can access the similar data concurrently. A database server also prevents from 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. The logical structures of an Oracle database include schema objects, data blocks, extents, segments and tablespaces. Technically, a schema can be defined as a compilation of database objects. A schema is typically owned by a database user and has the identical name as per that user. Schema objects are the logical structures and directly refer to data of the database system. Schema objects also include structures, such as tables, views and indexes. There is no relationship between a tablespace and a schema. Objects in the same schema can refer to different tablespaces and a tablespace can hold objects from different schemas.

An Oracle server consists of an Oracle database and an Oracle server instance. Every time a database is started, a System Global Area (SGA) is allocated and Oracle background processes are started. The combination of the background processes and memory buffers is called an Oracle **instance**. The Oracle Database is an Object Relational Database Management System (ORDBMS) produced and marketed by Oracle Corporation. The purpose of a database is to store and retrieve related information. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

The Oracle RDBMS stores data logically in the form of tablespaces and physically in the form of data files termed as 'datafiles'. Tablespaces can contain various types of memory segments, such as Data Segments, Index Segments, etc. Segments in turn comprise one or more extents. Extents comprise groups of contiguous data blocks. Data blocks form the basic units of data storage.

NOTES

NOTES

Check Your Progress

1. What is PL/SQL?
2. What are the advantages of PL/SQL?
3. Name the sections the PL/SQL program is divided into.

3.3 DATA TYPES IN PL/SQL

A program has many inputs and outputs in the form of variables and constants. These variables and constants specify the storage format, type of value and a range of the value that could be stored. PL/ SQL provides various data types which are system defined and also give the flexibility to the programmer to create their own data types which fit in the business needs.

Classification of Data Types

Data types are classified into:

- Scalar data types
- Composite data types

- **Scalar Data Types**

Scalar data types are the predefined data types. Scalar data type does not have internal components that can be manipulated individually.

Scalar data types are classified into four categories:

- o Number
- o Character
- o Boolean
- o Date and time

- **Composite Data Types**

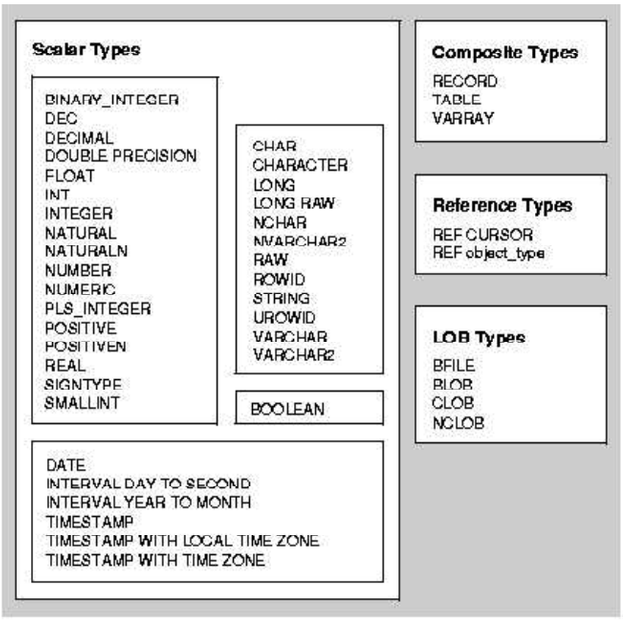
Composite data types are the user-defined data types. A composite data type has internal components that can be manipulated individually. These data types are created by the user to fulfill the business needs. These data types provide the flexibility, increase reliability and improve readability to a PL/ SQL programmer.

Composite data types are classified into the following categories :

- o Record
- o Table
- o Varray

The following screen shows the scalar and composite data types.

NOTES



3.4 LITERALS AND COMMENTS IN PL/SQL

Literals are the smallest unit of any program. There are different types of literals: Some of them are described as follows:

- **Number literals:** Number literals represent the whole or real numbers in PL/ SQL. These literals can have up to 38 digits and could be either positive or negative numbers.

  Following are the examples of number literals:

  +245

  323

  –25

  25e –04

  56.002

- **Text literals:** Text literals represent the character value in PL/ SQL. Text literals are blocked within the single quotes (‘’). Text literals may contain alphanumeric values.

  Following are the examples of text literals :

  ‘A-09’

  ‘PL/ SQL’

  ’28-MAY-03'

  ‘T’

- **Integer literals:** Integer literals represent the whole numbers in PL/ SQL and are up to 38 digits. Integer literals can be either positive numbers or negative numbers. If you do not specify a sign, then a positive number is assumed. Following are some of the examples of valid integer literals :

NOTES

23

+ 23

– 23

Number literals can be up to 38 digits. Number literals can be either positive or negative numbers. If you do not specify a sign, then a positive number is assumed. Following are some of the examples of valid number literals :

25

+ 25

– 25

25e–04

25.607

3.4.1 Comments in PL/ SQL

In Oracle, comments may be introduced either for single line or for multiple lines.

Types of Comments

1. `/*...*/` is used for multiple line comments.
2. – is used for single line comments.

The example for single line comment is given as follows:

```
– This is a PL/ SQL program to calculate employee salary
Declare
...
```

The example for multiple line comment is given as follows :

```
/*
This is a PL/ SQL program.
It calculates employee salary.
*/
Declare
...
```

Check Your Progress

4. How are the data types classified?
5. What is a composite data type?
6. What is number literal?

3.5 VARIABLES IN PL/SQL

Variables are the identifiers of data type. These variables could be the identifiers of either system defined (scalar) data types or the identifiers of user-defined (composite) data type, i.e. record, table or varray.

Important Points to Remember

- Name of the variables must start with a character.
- Name of the variable must not have space or any special symbol.
- Name of every variable must be unique.
- Value to a variable could be assigned during its declaration.
- Variable declaration can be of any data type.

Example:

```
Name char (30 ) ;
Salary Number (8, 2 ) ;
Date_of_join Date ;
```

- Constants can be of any data type:

Example:

```
Pi constant number ( 3, 2 ) := 3.5 ;
Status Booleans := TRUE ;
```

Pi and status are assigned values during declaration, which makes them constant.

3.5.1 Example of PL/ SQL Program

The following program calculates the sum of two numbers.

```
DECLARE
   number_1     NUMBER (10) ;
   number_2     NUMBER (10) ;
   res NUMBER (10)  ;

BEGIN
   number_1 := &number_1 ;
   number_2 := &number_2 ;
   res := number_1 + number_2 ;
DBMS_OUTPUT.PUT_LINE ( 'Sum is ' || res ) ;
END ;
/
```

Description of the preceding program is given as follows:

In declaration section three variables are declared and named, `number_1`, `number_2` and `res` of number data type.

In executable section value in `number_1` and `number_2` variables are taken by the user interactively. Here `&` symbol prompts the user to enter the value and **:=** (assignment operator) is used to assign value to variables.

Value for `'res'` variable is calculated to produce the sum of `number_1` and `number_2.`

DBMS_OUTPUT.PUT_LINE is used to display the output of a program.

NOTES

NOTES

3.6 PACKAGE FUNCTION AND PROCEDURES

A package is a database object. It is a collection of various objects as, procedures, functions, cursors, variables and constants.

There are two types of packages :

1. Built-in packages
2. User defined packages

1. **Built-in packages:** Built-in packages such as `DBMS_OUTPUT`, `DBMS_SQL`, `DBMS_DDL`, `DBMS_TRANSACTION`, etc. caters pre-defined functionality.
2. **User-defined packages:** User defined package serve the user as, per the changed business needs.

A package consists of two parts :

- Package specification
- Package body

• **Package specification:** In package specification one could declare variables, constants, exceptions, cursors, sub-procedures and other database objects.

Package specification could be created by using the CREATE PACKAGE statement.

The Syntax *to* create package specification is as follows :

```
CREATE [ or Replace ] Package < package_name > IS <
declarations >
Begin
( Executable statements )
END <package_name > ;
```

The sub-procedures declared in package specification must be declared in package body.

• **Package body :** The actual implementation of declared sub-procedures and cursors is done in package body.

The Syntax for the CREATE BODY statement is as follows :

```
CREATE [ or Replace ] package  < package_name > IS <
declarations >
Procedure < procedure_name > ( variable  data type) ;
Function < function_name > ( variable  data type ) return
data type ;
END < body_name > ;
```

A Package Function

Following example declares a function `getGrade` which would accept an argument of varchar data type and would return a value of varchar data type.

Step-1

```
Create or replace package pkg_marksheet
```

```
is
        Function getGrade ( rno varchar ) return varchar
;
End pkg_marksheet ;
/
```

NOTES

The above code would create a package with the name pkg_marksheet. This package contains a function named getGrade. This function will accept an argument of varchar type and will return a value of varchar type.

Save the program with the name pkg_marksheet and then compile it by using :

```
SQL > @ pkg_marksheet ;
```

The output of the above PL/ SQL code when compiled will be given as follows:

```
Package created.
```

Step-2

The function pkg_marksheet is declared in package body shown as follows :

```
    return 'B-' ;
elsif per >= 30 then
    return 'C' ;
else
    return 'F' ;
end if ;
end getgrade ;
end pkg_marksheet ;
/
    return 'B-' ;
elsif per >= 30 then
    return 'C' ;
else
    return 'F' ;
end if ;
end getgrade ;
end pkg_marksheet ;
/
```

Save the program with any name (for example marksheet) and then compile it by using :

```
SQL> @ marksheet ;
```

The output of the preceding PL/ SQL code when compiled is given as follows:

```
Package body created.
```

NOTES

Calling Package Function

To call the function declared in package specification the reference of package name need to give as be given is as follows:

The Syntax to call a package function is as follows :

```
Package_name.function_name ;
```

The example to call a package function is as follows :

```
pkg_marksheet.getGrade ('A-08-12') ;
```

Where pkg_marksheet is a package name in which a function `getGrade` is declared which takes a varchar argument A-08-12.

A Package Procedure

Following example declares a procedure `show_book_price` which would accept an argument of varchar data type.

Step-1

```
Create or replace package book_price IS
      procedure show_book_price ( bcode varchar ) ;
End book_price ;
/
```

The code would create a package with the name `book_price.` This package contains a procedure named `show_book_price.` This procedure will accept an argument of varchar type.

* Procedure can not return any value.

Save the above program with the name book_price and then compile it by using :

```
SQL > @ book_price ;
```

The output of the above PL/ SQL code when compiled is given as follows:

Package created.

Step-2

```
create or replace package book_price as

    procedure show_book_price ( bcode varchar )
IS

p number ( 7 , 2 ) ;

begin
    select price into p from book where b_code = bcode ;
    dbms_output.put_line ( 'Book Price is ' || p ) ;

end show_book_price ;
end book_price ;
```

Save the program with the any name (for example, `suppose show_price`) and then compile it by using :

```
SQL > @ show_price ;
```

NOTES

The output of the above PL/ SQL code when compiled is given as follows:

```
Package body created.
Calling Package Procedure
```

To call the procedure declared in package specification the reference of package name need to be given is as follows:

The syntax to call a package procedure is as follows :

```
Package_name.procedure_name ;
```

The example to call a package procedure is as follows :

```
book_price. show_book_price ( 'B003' ) ;
```

Where `book_price` is a package name in which a procedure `show_book_price` is declared which takes a varchar argument B003.

Check Your Progress

7. What is a variable?
8. What is a package?
9. Name the two types of packages?

3.7 ERROR HANDLING IN PL/SQL

In PL/ SQL error is called exception. Error may occur due to various reasons such as coding mistakes, hardware failure, system resource problems and many other reasons. Due to these errors program terminates abnormally.

Whenever an errors which is an exception handler is raised the normal program execution stops, and the control is transferred to the exception handler of the PL/ SQL program block and the error is handled.

Type of Exception :

1. Internal exception
2. User-defined exceptions

1. **Internal exception:** An internal exception is raised implicitly by the system when PL/ SQL program violates any Oracle rule. For example, you try to store data in a variable more than the range of the specified data type, try to select records from a table which does not exist and many more.
2. **User-defined exceptions :** Programmer can define their own exceptions to maintain data security, consistency and integrity as per the organization rules.

Table 3.1 shows the list of internal exceptions.

NOTES

Table 3.1 *Internal Exceptions*

Exception	Explanation
ZERO_DIVIDE	This exception is raised when PL/ SQL program attempts to divide a number by zero.
NO_DATA_FOUND	This exception is raised when SELECT INTO statement returns no rows while expected to return.
CURSOR_ALREADY_OPEN	This exception is raised when you try to open a cursor which is already open.
INVALID_NUMBER	This exception is raised when, the conversion of a string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	This exception is raised when PL/ SQL program attempts to log on to Oracle with an invalid username and/or password.
NOT_LOGGED_ON	This exception is raised when PL/ SQL program issues a database call without being connected to Oracle.
STORAGE_ERROR	This exception is raised when PL/ SQL runs out of memory.
TOO_MANY_ROWS	A select into statement returns more than one row while expected only one.
VALUE_ERROR	This exception is raised when data type or data size is invalid.
PROGRAM_ERROR	This exception is raised when PL/ SQL has an internal problem.
OTHERS	This exception is raised when error is unknown or not explicitly defined.

Other than the exceptions mentioned in Table 3.1, there are various other exceptions. Some of them are as follows:

- DUP_VAL_ON_INDEX
- TIMEOUT_ON_RESOURCES
- INVALID_CURSOR
- SYS_INVALID_ROWID
- SUBSCRIPT_OUTSIDE_LIMIT
- SUBSCRIPT_BEYOND_COUNT

Example :

```
Declare
    b_title varchar (40) ;
```

```
Begin
    b_title := '&b_title' ;
    Select title into b_title from book where title =
b_title ;

Exception
    when NO_DATA_FOUND then
    dbms_output.put_line ( 'No Record Found' ) ;

    when TOO_MANY_ROWS then
    dbms_output.put_line ( 'Query Returns More Than One
Query' ) ;
End ;
/
```

In the above program select query is used to select book title into variable `B_title`. Two internal exceptions are handled named `NO_DATA_FOUND` and `TOO_MANY_ROWS`. If query returns more than one records then `TOO_MANY_ROWS` exception would be raised by the system, if no record matches then `NO_DATA_FOUND` exception would be raised.

User Named Exception Handlers

There are many system exception for which Oracle does not provide a name. These known as unnamed system exceptions. But every exception is provided a number by Oracle. A name could be assigned to these unnamed exceptions by using `Pragma Exception_Init ()`.

You could assign a name to the unnamed system exception by using a `Pragma` called `Exception_Init` shown as follows:

```
Pragma Exception_Init ( exception_name , Oracle error
number ) ;
```

In the above example exception name is the user defined name of the exception that will be associated with Oracle error number.

Syntax :

```
DECLARE
   exception_name EXCEPTION ;
    PRAGMA EXCEPTION_INIT ( exception_name , Err_code
) ;
BEGIN
   Executable statement ;
   . . .
EXCEPTION
    WHEN exception_name THEN
    Handle the exception
END ;
```

Example :

```
DECLARE
    child_record_exception EXCEPTION ;
    PRAGMA EXCEPTION_INIT ( child_record_exception, -2292
) ;
```

NOTES

```
BEGIN
    Delete from course where C_code = 'PG001' ;
EXCEPTION
    when child_record_exception then
    DBMS_OUTPUT.PUT_LINE ( 'Child Record Present, Can not
delete this record' );
End ;
/
RAISE_APPLICATION_ERROR  ( )
```

A user can assign an error message by using `Raise_application_error ( )` to make the error message more descriptive for the end-user. `Raise_application_error ( )` is a build-in procedure.

Syntax :

```
Raise_application_error ( Oracle error number, user defined
error message ) ;
```

The error number whose range is in between -20000 and -20999 is given in Oracle error number and a user-defined error message is defined in user defined error message parameter.

User-defined Exceptions

Other than the pre-defined exceptions you could define your own exception to validate data against business requirements. For example if user wants to update total marks of student but subject marks are `NULL` an error must be raised by the system to alert the user.

A user defined exception must be declared within the declaration section by the keyword **EXCEPTION** and must be raised explicitly by **RAISE** statement within the executable section.

Example :

```
DECLARE
    null_marks EXCEPTION ;
    rno   Number (3) ;
    s1 Number (3) ;
    s2 Number (3) ;
    s3 Number (3) ;
    s4 Number (3) ;
BEGIN
    Rno := &rno ;
          Select sub1, sub2, sub3, sub4 into s1, s2, s3
, s4 from marks where roll_no = rno ;
    If s1 is NULL or s2 is NULL or s3 is NULL or s4 is NULL
then
    RAISE null_marks ;
    End if ;
    Update marks set total = s1 + s2 + s3 + s4 where
roll_no = rno ;
EXCEPTION
    WHEN  null_marks  THEN
    DBMS_OUTPUT.PUT_LINE ( 'Subject marks are NULL' ) ;
```

```
END ;
/
```

In the above example `null_marks` is a user defined exception which must be raised explicitly by using `RAISE` statement. This exception would be raised when marks in any subject would be `NULL`.

NOTES

3.7.1 Oracle Transactions

All the changes that you make through data manipulation language (DML) command are known as transaction. A transaction is a logical group of work. Transactions that you do on a database temporarily stored on the client machine. They can either be made permanent or could be canceled by the user.

Oracle provides few commands to control the transactions given as follows :

1. Commit
2. Savepoint
3. Rollback

1. Commit: The commit command is used to make the transaction permanent to the database. The commit command ends the current transactions.

Important Points to Remember

- All the transaction done by the user becomes permanent by this command.
- After committing the records updated database can be seen by other users as well.
- If any locks are acquired by the transaction they will released after committing the records.
- Once a transaction is committed it can not be roll backed.

Example:

```
SQL > Commit to work ;
```

The keyword work is optional which is used to increased the readability, you could also write

```
SQL > Commit ;
```

When commit command is executed Oracle prompts a message shown as follows:

```
Commit complete.
```

* When you exit from the Oracle normally, **AUTOCOMMIT** command gets executed implicitly to save all the changes to the database.

```
DECLARE
    null_marks EXCEPTION ;
    rno   Number (3) ;
    s1 Number (3) ;
    s2 Number (3) ;
    s3 Number (3) ;
    s4 Number (3) ;
BEGIN
```

NOTES

```
    Rno := &rno ;
           Select sub1, sub2, sub3, sub4 into s1 , s2 ,
s3 ,s4 from marks where roll_no = rno ;
    If s1 is NULL or s2 is NULL or s3 is NULL  or s4 is
NULL then
    RAISE null_marks ;
    End if ;
    Update marks set total = s1 + s2 + s3 + s4 where
roll_no = rno ;
       COMMIT ;
EXCEPTION
    WHEN null_marks THEN
    DBMS_OUTPUT.PUT_LINE ( 'Subject marks are NULL' ) ;
END ;
/
```

2. Rollback : The rollback command is used to terminate the current transaction. All the changes made to the rollback database can be undone by rollback. It is generally used when a session disconnects from the database without completing the current transaction.

Example :

```
SQL > Rollback work ;
```

The keyword work is optional which is used to increased the readability, you could also write

```
SQL > rollback ;
```

When commit command is executed Oracle prompts a message shown as follows :

```
Rollback complete.
```

* Rollback undoes the whole transaction made after the last committed transaction.

3. Savepoint: Unlike rollback savepoints are used to terminate a specified set of transaction. Savepoints are used to make sets of logically related transaction. These savepoints could be used to rollback a specified set of transaction.

Some important points to remember are:

- Savepoint is useful during complicated transaction. If any part of transaction fails it can be rollbacked to maintain data inconsistency and integrity.
- Any lock acquired by the SQL statements releases after savepoint is used.
- Any set of transaction can be undone by using savepoint.
- The transaction is not finished, until SQL statements are still pending.

Example :

```
SQL > Savepoint book_insert ;
SQL > Insert into book ( b_code , title , price ) values
( 'B007', 'RDBMS', 220 ) ;
SQL > Insert into book ( b_code,  title, price) values
('B008', 'Oracle', 20 ) ;
```

```
SQL > Savepoint book_delete ;
    SQL > delete from book where b_Code = 'B007' ;
SQL > Rollback to savepoint book_delete ;
```

NOTES

In the preceding example two savepoints are created, names `book_insert` and `book_delete`, if you rollback up to the savepoint `book_delete` only transaction where book has been deleted would be undone and the rest will remain the same and you could either rollback or commit.

Check Your Progress

10. Give some reasons for the occurrence of errors in PL/SQL.
11. Name the various types of exceptions.
12. What is a transaction?

3.8 STORED PROCEDURES

A procedure is a subprogram that performs a specific action. The input in procedures is given as arguments. The procedure manipulates these arguments and produces some output. It cannot return any value.

Create a Procedure

A procedure could be written in any text editor such as Notepad, Write pad, and even in MS-Word. The default editor of Oracle is Notepad.

To invoke the editor, write ED and procedure name at SQL prompt as follows:

```
SQL> ED procedure_name
```

The **Syntax** to create a procedure is as follows:

```
CREATE [OR REPLACE] PROCEDURE proc_name (list of parameters]
IS
  Declaration section
BEGIN
   Execution section
EXCEPTION
  Exception section
END;
```

Description of the Syntax

CREATE Procedure

CREATE is used to create a procedure. If no other procedure with the given name exists or if any other procedure with the same name exists, it would be replaced with the new code.

OR REPLACE Procedure

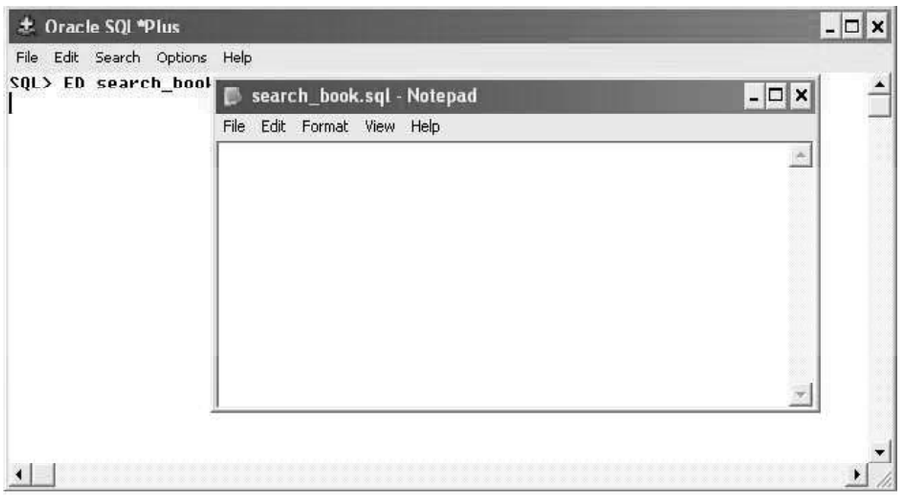
OR REPLACE is used to re-create the procedure if it already exists. Use this clause to change the definition of an existing procedure without dropping or re-creating the procedure.

NOTES

Users who had previously been granted privileges on a redefined function still would have the access of the function without being granted the privileges again.

IS — It is similar to DECLARE in PL/SQL Blocks. Variables could be declared between IS and BEGIN.

For example, the given screenshot is an example of a procedure search_book in Oracle editor. To invoke the editor write ED and procedure name as shown. Notepad would open. Now write the codeon the notepad and save the file. After writing the code exit from the editor.



Consider table 3.1 which contains the details of books of a library. A stored procedure is written to show the details of books. The book code is passed as a parameter in this procedure.

Table 3.2 *Book*

B_CODE	TITLE	AUTHOR	PRICE
10001	Sales and Marketing	Deborah	300
10002	Information Technology and its applications	Michael	600
10005	Management information system	Parker	230
10006	Product, Price and Promotion	Deborah	234
10007	Learn your self : Database	Wood	234
10009	Database Management System	Wood	142

Procedure: search_book

```
CREATE PROCEDURE search_book (code IN NUMBER) IS
    B_author varchar ( 20 );
    B_title varchar (20) ;
    B_pice Number(5);
BEGIN
Select title,author,price into b_title,b_author,b_price
```

```
Where b_code = code ;
DBMS_OUTPUT.PUT_LINE('Book Title is '|| b_ title);
DBMS_OUTPUT.PUT_LINE ( 'Book Author is ' || b_author ) ;
DBMS_OUTPUT.PUT_LINE ( 'Book Price is ' || b_price ) ;
END;
/
```

The above procedure, could be used to display book detail of any given book code. The next step is to compile this procedure.

Compile Procedure

To execute any stored procedure, it is necessary to compile it. To compile a procedure the following command is used:

```
Syntax:
        SQL> @ procedure_name ;
Example:
        SQL> @ search_book ;
```

If the procedure does not contain any error then the system would prompt a message as follows:

```
Procedure created.
```

Some times there may be a mistake in a procedure that you create. In this case, the system would prompt a message as follows:

```
Warning: Procedure created with compilation errors.
```

Show Error:

As you get a warning message after compiling the procedure that is completely inadequate, for debugging the problem you can see the error messages. Use `Show Errors` command to check the error in a stored procedure as follows:

```
SQL> Show error
```

Execute Procedure

The above procedure could be executed by using `Exec` command:

```
SQL> Exec search_book (10006) ;
```

The above command will execute the procedure search_book and prompt a message as follows:

```
Procedure successfully completed.
```

If a procedure is completing successfully, you will not see the output of the procedure. The reason behind this is that by default the console output remains on the off mode. To turn on the console, the output uses the following command:

```
SQL>Set Serveroutput On
```

`SET SERVEROUTPUT ON` is the SQL*Plus command. This command is used to activate the console output. You could also set the server output off by writing the command:

```
SQL>Set Serveroutput Off
```

This procedure could also be called inside other procedures (or functions). Calling a procedure inside another PL/SQL program increases the reusability of program code and improves maintainability.

NOTES

Executed Inside Another PL/SQL Program: The above procedure could be executed inside another PL/SQL program by writing the procedure name as follows:

```
BEGIN
Search_book ('B006' ) ;
END;
/
Executed Inside Another Procedure:
```

For example:

```
Create or replace procedure main_proc (name in varchar,
age in number) IS
begin
dbms_output.put_line ( name || ' is ' || age || ' years
old' ) ;
end;
/
```

The above procedure named `main_proc` has two arguments: name and age of data type varchar and number, respectively.

To call this procedure within another procedure the code is as follows:

```
Create or replace procedure sub_proc (name  varchar,
date_of_birth date)
IS
    age number;
begin
age := round ( months_between ( sysdate, date_of_birth )
/ 12 ) ;
    main_proc ( name, age ) ;
end;
/
```

The above procedure named `sub_proc` has two arguments: name and date_of_birth of data type varchar and date, respectively.

`Round ()` is a function which returns a number rounded to a specified number of decimal places.

`Months_between()` is a function which returns the number of months between date1 and date2. In the given procedure it return the number of months between sysdate (current date) and date passed as an argument.

`Sysdate` returns the current system date and time on your local database.

```
Execute the Sub-Procedure
```

The above procedure could be executed by using the `Exec` command:

```
SQL> Exec second_proc ('Deep', '10-nov-1975' ) ;
```

NOTES

Parameters in Stored Procedures: Parameters are used to pass information back and forth between the procedures and functions and calling PL/SQL block. These are the variables that are available to a stored procedure to manipulate the data. Parameters are declared when a procedure or function is declared.

Parameter names follow the Oracle naming convention.

The parameters of a stored procedure have three major attributes as shown in Table 3.3.

Table 3.3 *Attributes of Procedure Parameters*

Parameter Attribute	Description
Name	Name is the parameter name which is an identifier of a data type. This parameter could be the identifier of any system defined (scalar) data types.
Mode	This indicates whether the parameter is an input-only parameter (IN), an output-only parameter (OUT), or is both an input and an output parameter (IN OUT). If the mode is not specified, then IN is assumed.
Data Type	This is a standard PL/SQL data type.

Parameter Modes

Parameter in stored procedures could be defined in various modes to determine how the program can use and manipulate the value assigned to the formal parameter.

There are three different modes of parameters which are as follows:

1) IN Mode
2) OUT Mode
3) IN OUT Mode

The IN Parameter

This is similar to passing parameters in programming languages. We can pass values to the stored procedure through these parameters or variables. This parameter is a `Read Only` parameter whose value could be read to display or to insert and update in a table. However, the value of this parameter cannot be changed inside the procedure.

In a procedure IN is the `Default` mode for parameters. If parameter mode is not specified by the programmer, then the parameter mode is automatically considered an IN parameter mode.

The actual parameter for an IN parameter can be a variable, a named constant, a literal, or an expression.

The syntax to pass an IN parameter is as follows:

```
CREATE [OR REPLACE ] PROCEDURE  < procedure name >  (
param_name IN data type, param_name IN data type,  ... )
```

The example to pass an IN parameter is as follows:

```
CREATE  [OR  REPLACE]  PROCEDURE  proc_parameter
```

NOTES

```
( issue_date IN date, B_title varchar2, return_date
IN date )
Begin
…
Exception
…
End;
/
```

In the given syntax, the procedure `proc_parameter` is the procedure name, `issue_date, B_title,` and `return_date` are the parameters and the mode of all three parameters is `IN` which makes the parameters read only. The mode for `B_title` is not defined. Therefore, by default it will also be an IN parameter.

The OUT Parameter

This is a Write Only parameter. This parameter is used to send the OUTPUT from a procedure or a function. The value cannot be passed to OUT parameters while executing the stored procedure from SQL prompt or from any other procedure or PL/ SQL progam. The value to the OUT parameter can be assigned inside the stored procedure and the calling program can receive this output value. The OUT parameters have restricted usage.

The Syntax to pass an OUT parameter is as follows:

```
CREATE [OR REPLACE] PROCEDURE <procedure name>
param_name OUT data type, param_name OUT data type, ...)
```

The example to pass an OUT parameter is as follows:

```
CREATE [OR REPLACE] PROCEDURE proc_parameter ( issue_date
OUT date, B_title IN varchar2, return_date OUT date
) IS
Begin
…
Exception
…
End;
/
```

In the above example procedure `proc_parameter` is the procedure name, `issue_date, B_title, and return_date` are the parameters. The mode of `issue_date` and return_date parameters is `OUT` which makes the parameters read only, whereas the mode of b_title parameter is IN. The parameter should be explicitly declared as OUT parameter.

Important points to remember are as follows:

- You cannot assign an OUT parameter's value to another variable or even use it in a re-assignment to itself. An OUT parameter can be found only on the left side of an assignment operation.
- The value of an OUT parameter cannot be assigned to any other variable.

NOTES

- The value of an OUT parameter cannot be re-assigned.
- An OUT parameter can be used only on the left side of an assignment operation.
- The default value to an OUT parameter cannot be assigned. Its value can only be assigned inside the body of the stored procedure.
- An OUT parameter must be a variable. It cannot be a constant, literal or an expression.
- The OUT parameter provides a level of security due to its read only feature.

The IN OUT Parameter

The IN OUT parameter has the properties of both IN and OUT mode. With the IN OUT parameter, value could be passed into the program and values could also be returned back to the calling procedure or PL/ SQL program.

Important points to remember are as follows:

- An IN OUT parameter cannot have a default value.
- IN OUT parameters have the properties of both IN and OUT parameters
- An IN OUT parameter must be a variable. It cannot be a constant, literal or expression.
- You can use the IN OUT parameter in both sides of an assignment

The syntax to pass an IN OUT parameter is as follows:

```
CREATE [ OR REPLACE ] PROCEDURE  < procedure name >
(param_name IN OUT data type, param_name IN OUT data
type,  ... )
```

The example to pass an IN OUT parameter is as follows:

```
CREATE [OR REPLACE] PROCEDURE proc_parameter
( issue_date  IN OUT  date,  B_title  IN OUT varchar2,
return_date  IN OUT  date ) IS
Begin
…
Exception
…
End;
/
```

In the given syntax, the procedure `proc_parameter` is the procedure name, issue_date, B_title and return_date are the parameters and have the same IN OUT mode.

Defining Parameter with Different Modes in a Procedure

The parameter mode is defined immediately after the parameter name and before the parameter's data type. The IN is the default parameter mode.

The syntax to define Parameter Mode is as follows:

```
CREATE [ OR REPLACE ] PROCEDURE <procedure name>   ( <
parameter name >  parameter mode Data Type,  < parameter
name >  parameter mode Data Type,  … )
```

NOTES

The example to define Parameter Mode:

```
CREATE  [ OR REPLACE ] PROCEDURE proc_parameter     (
issue_date_in  IN  date,  B_title  IN OUT  varchar2,
return_date  OUT  date ) is
Begin
…
Exception
…
End;
/
```

The `proc_parameter` procedure takes in two pieces of information: The books issue date and the book title. It then returns two pieces of information: book title and return date.

Example: Procedure for XYZ Company

XYZ company is manufacturing computer hardware parts at different locations of the country. The database is being shared by all the branches of the company. The company has decided to increment the salary as per the following conditions:

1. To those employees working in department number 30 with designation as MANAGER the company has decided to give `15 percent` hike in salary.
2. To those employees working in department number 30 with designation SALESMAN the company has decided to give `10 percent` hike in salary.
3. For the rest of the employees the company has decided to give `8 percent` hike in salary.

   The records are shown in Table 3.4

Table 3.4 *Employee*

EMP_CODE	E_NAME	DESIGNATION	SALARY	DEPTNO
7369	SMITH	CLERK	15000	20
7499	ALLEN	SALESMAN	35000	30
7521	WARD	SALESMAN	32000	30
7566	JONES	MANAGER	55000	20
7654	MARTIN	SALESMAN	30000	30
7698	BLAKE	MANAGER	60000	30
7782	CLARK	MANAGER	64000	10
7788	SCOTT	ANALYST	58000	20
7839	KING	PRESIDENT	70040	10
7844	TURNER	SALESMAN	30430	30
7876	ADAMS	CLERK	23000	20

To update the employee's salary with the interactive SQL is tedious task. This task could be performed by writing a procedure as follows:

NOTES

```
/* Procedure emp_sal to update the employee salary */
create or replace procedure emp_sal ( EID number ) IS
E_dept varchar2 ( 10 ) ;
E_desg varchar2(9);
E_sal number ( 8 , 2 ) ;
begin
select salary , deptno, designation into
E_sal,E_dept,E_desg from emp where emp_code = EID ;
—dbms_output.put_line ( E_sal ) ;
    if E_dept = 30 and E_desg = 'MANAGER' then
       e_sal := e_sal +  ( E_sal * 15 ) / 100 ;
elsif E_dept = 30 and E_desg = 'SALESMAN' then
       e_sal := e_sal + ( E_sal * 10 ) / 100 ;
else
       e_sal := e_sal + ( E_sal * 8 ) / 100 ;
end if;
—Update salary into employee table
update emp set salary = e_sal where emp_code = eid ;
commit;
end;
/
```

The above procedure emp_sal has a parameter of number type to accept the employee ID. The salary, designation and department would be selected of the given employee ID to check the criteria for increment as mentioned. The salary would be updated in the employee table.

Compile the Procedure

Compile the procedure as follows:

```
SQL>  @ emp_sal ;
Execute the Procedure
```

Execute the procedure as follows:

```
SQL> Exec emp_sal (7782) ;
```

Dropping Procedures

If any procedure is no more required, you can drop it by using DDL command DROP.

The syntax to DROP a procedure is as follows:

```
DROP PROCEDURE procedure_name ;
```

For example,

```
SQL> DROP PROCEDURE search_book ;
```

The above command will drop the procedure named search_book and will prompt a message as follows:

```
Procedure dropped.
```

NOTES

3.9 STORED FUNCTIONS

A stored function always returns a result and can be called inside an SQL statement just like ordinary SQL function. A function parameter is the equivalent of the IN procedure parameter, as functionals use the RETURN keyword to determine what is passed back.

User-defined functions or stored functions are the stored procedures which have the features of all procedures. They can accept parameters, perform calculations based on data retrieved and return the result to the calling SQL statement, procedure, function or PL/SQL program. A function returns a value.

Create a Function

The **syntax** to create a function is as follows:

```
CREATE OR REPLACE FUNCTION function_name ( function_params
)
RETURN return_type IS
      Declaration statements
BEGIN
      Executable statements
     RETURN something_of_return_type ;
EXCEPTION
     Exception section
END;
```

Description of the Syntax

CREATE Function

This is used to create a function, if no other function with the given name exists.

OR REPLACE Function

OR REPLACE is used to re-create the function if the given function name already exists. If no function exists with the given name, it creates the new function. You can also use OR REPLACE clause to change the definition of an existing function without dropping, re-creating and regranting privileges previously granted on the function to other users. If you redefine a function, then Oracle Database recompiles it.

IS—It is similar to DECLARE in PL/SQL Blocks. Variables could be declared between IS and BEGIN.

RETURN Clause

Function returns a value. The RETURN clause is used to specify the data type of the return value of the function. Since every function must return a value, this clause is mandatory to use. The return value can have any data type supported by PL/SQL.

Example: Functions can be very useful in many situations. For example,

functions can be useful when you need to calculate the total monthly sale in different areas and of different items. Or you want to calculate the expenses of an organization. In such instances functions are useful.

Consider Table 4.5, which contains the detailed of accounts of account holders of bank.

Table 3.5 *Account_holder*

ACC_NO	NAME	TYPE_OF_AC	CONTACT_NO	AC_BALANCE
120040	Tom	Saving	98978800	15620
120040	Merlisa	Saving	98981600	26500
120041	George	Saving	8787700	16560
120041	Smith	Saving	6050234	25500
120042	Loise	Current	6050234	26660
120043	marry	Current	38042342	70080

A stored function is given to return the balance of an account holder. The account number is passed as a parameter in this function.

Function: `get_balance ()`

```
/* This is a stored function which returns the total
balance of all saving accounts*/
CREATE or replace FUNCTION get_balance ( no IN NUMBER)
RETURN NUMBER
IS acc_bal NUMBER ( 11 , 2 ) ;
BEGIN
SELECT sum ( ac_balance ) INTO acc_bal  from account_holder
WHERE acc_no = no ;
RETURN ( acc_bal ) ;
END;
/
```

The given function, `get_balance ()` has a parameter of number type to accept the account holder's account number. The `acc_bal` is a variable in which the balance of the given accout holder is stored and returned to the caller program.

Save the above file with the name `account_balance.SQL`

Compile Function

To execute any stored procedure it is necessary to compile it. To compile a procedure the following command is used:

The syntax is as follows:

```
SQL> @ function_name ;
```

For example,

```
SQL> @ account_balance ;
```

Execute Function

NOTES

The above function could be called inside other PL/SQL program or procedures. For example:

```
/* This is a PL/ SQL program which calls a stored function
get_balance () */
DECLARE
ac number(14);
amount number(10,2);
BEGIN
ac:=&ac;
amount := get_balance (ac) ;
dbms_output.put_line (' The Balance Amount of account '
|| ac || ' is ' || amount ) ;
END;
/
```

The function created in the preceding example can be used in a SQL statement. For example:

```
SQL> SELECT get_balance (120041) FROM DUAL ;
```

Dropping Function

If function is no more required, you can drop it by using DDL command DROP.

The **Syntax** to DROP a function is as follows :

```
DROP FUNCTION function_name ;
```

The **Example** to DROP a function is as follows.

```
SQL > DROP FUNCTION sum ;
```

The given command will drop the function named sum and will prompt a message which is as follows:

```
Function dropped.
```

Sometimes, the term stored procedure is used for both stored procedures and stored functions.

The differences between procedures and functions is as follows:

- A function always returns a value to the calling program or procedure
- A procedure cannot return a value to the calling program or procedure
- Procedures and functions can both return data in OUT and IN OUT parameters
- Functions can be called from SQL, procedure cannot
- Function is considered as an expression but procedure is not

Where do procedures and functions reside?

All the procedures and functions are stored in Oracle database. That is why they are called stored procedures. Before the procedures and functions are stored in the Oracle database, they are compiled by the Oracle engine automatically. These stored procedures could be called any number of times in PL/ SQL program, procedure or could be executed at the SQL prompt.

NOTES

3.10 ADVANTAGES OF STORED PROCEDURE AND FUNCTION

The various advantages of using stored procedures and functions are as follows:

Maintainability: Sub-programs are easy to maintain because they serve a specific function. If any change is required to be made to that function, the entire application need not be change.

Reusability: Once a procedure and function is written and compiled, it could be used in any number of programs.

Security: Stored procedures and functions can help enforce data security. Through procedures and functions, users can be restricted to perform limited database operations.

For example, you can grant permission to users to access a procedure that selects the records of a table. However, you cannot grant them the permission to update or delete records. When a user invokes the procedure, the procedure executes with the privileges of the procedure's owner.

Performance: Stored procedures and functions reduce the network traffic as compared to a number of individual SQL queries. As the procedure is already compiled and sent at one run to the database, no further compilations are done, which help in improving the performance.

Memory Allocation: If multiple users concurrently access any procedure or function, it is required to load it only as a single copy into shared memory.

Productivity: For the common operations of an application, procedures and functions are reused. The reused code increases the development productivity.

Error Reduction: By developing an application with various procedures and functions, the chances of errors reduces. This helps in maintaining data integrity.

CHECK YOUR PROGRESS

13. List the advantages of using stored procedure and function.
14. What is a procedure?
15. Define user-defined functions.
16. How can a function be dropped when it is not required?
17. Write at least two differences between procedures and functions.

3.11 ANSWERS TO 'CHECK YOUR PROGRESS'

1. PL/SQL is an acronym for procedural language/structural query language. It supports procedural features and SQL commands.
2. Some of the advantages of PL/SQL are as follows:
   - It gives better performance.

- It is easily portable from one place to another.
- It increases productivity.
- It helps in handling error.

NOTES

3. PL/SQL is divided into following three sections:
   - Declaration section
   - Execution section
   - Exception handling section
4. Data types are classified into the following:
   - Scalar data types
   - Composite data types
5. A composite data type contains internal components that can be manipulated individually.
6. A number literal represents the whole or real number in PL/SQL (procedural language/structured query language).
7. A variable is an identifier of data type.
8. A package is a collection of various database objects such as functions, cursors, variables, etc.
9. The two types of packages are as follows:
   - Built-in packages
   - User-defined packages
10. Errors occur due to the following reasons:
    - Coding mistakes
    - Hardware failure
    - System resource problems
11. The types of exception are as follows:
    - Internal exception
    - User-defined exception
12. All the changes that are made through data manipulation language (DML) commands are known as transaction.
13. The various advantages of using stored procedure and function are as follows:
    - Maintainability
    - Reusability
    - Security
    - Performance
    - Memory Allocation
    - Productivity
    - Productivity
14. A procedure is a subprogram that performs a specific action.

15. User-defined functions or stored functions are the stored procedures which have the features of all procedures
16. If a function is no more required, you can drop it by using DDL command DROP.
17. A function always returns a value to the calling program or procedure

    A procedure cannot return a value to the calling program or procedure

NOTES

3.12 SUMMARY

- Procedural language/structured query language (PL/SQL) is also known as an embedded SQL and is a superset of SQL.
- PL/ SQL supports procedural statements including control statements, loops, exception handling, structure query language and procedure and functions.
- A PL/ SQL program can be saved on a disk for further use and to increase the reusability of the code.
- The major advantages of using PL/ SQL are better performance, portability, increased productivity, error handling, object- oriented programming support and security.
- Every PL/ SQL statement should be terminated with a semicolon (;) .
- PL/ SQL program block is divided into three sections: declaration, execution and exception handling.
- In PL/ SQL, data types are classified as scalar and composite data types.
- Scalar data types are classified into four categories: number, character, boolean and date and time.
- Composite data types are classified into the following categories:
  - o Record
  - o Table
  - o Varray
- Literals are the smallest unit of any program. There are various types of literals such as numeric, integer and text.
- Variables are the identifiers of data type.
- Name of a variable must start with a character.
- DBMS_OUTPUT.PUT_LINE is used to display the output of a program.
- A package is a database object and is a collection of various database objects as procedures, functions, cursors, variables and constants.
- There are two types of packages: built-in and user-defined.
- A package consists of two parts, i.e. package specification and package body.

NOTES

- The sub-procedures declared in package specification must be declared in package body.
- In PL/ SQL, errors could be handled in exception handling block. There are two type of exceptions: internal and user-defined.
- All the changes made to the database through the DML command are known as transaction.
- Transactions that you do on a database are temporarily stored on the client machine. They can either be made permanent or can be canceled by the user.
- Oracle provides certain commands to control the transactions. These are Commit, Savepoint and Rollback.
- Oracle normally uses AUTOCOMMIT command to get execute implicitly to save all the changes on the database.
- There are two type of subprograms namely, stored procedure and stored function.
- Procedures and functions are compiled PL/SQL code block, stored in the data dictionary.
- Stored procedures increase the reusability, security, maintainability and performance as could be called in any number of programs.
- Subprograms are named PL/ SQL blocks that can take parameters as an input and can be invoked whenever required from other PL/ SQL program, subprogram, and SQL prompt.
- Sub-programs can have three parts: a declarative part, an executable part and an exception handling part.
- Procedures are generally used to perform specific tasks, whereas functions are used to compute a value.
- There are three modes of parameters, namely, IN mode, OUT mode and IN OUT mode.
- To pass values to a subprogram, the parameter mode IN is used and to return values, the OUT parameter mode is used.
- The IN OUT parameter is used to pass initial values to the subprogram when invoked and the parameter can also return updated values.

3.13 KEY TERMS

- **Package:** It is a database object.
- **Literal:** It is the smallest unit of any program.
- **Text literal:** It represents the character value in PL/SQL.
- **Integer literal:** It represents whole numbers in PL/SQL.
- **Rollback:** It is used to terminate the current transaction.
- **Procedure:** It is a subprogram that performs a specific action.

- **OR REPLACE:** It is a procedure that is used to re-create an already existing procedure.
- **Round():** It is a function which returns a number rounded to a specified number of decimal places.

3.14 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is PL/ SQL ? What are its advantages?
2. Write in brief about Oracle Database Service.
3. What is the difference between SQL and PL/ SQL?
4. State various sections of PL/ SQL program.
5. Mention the data types available in PL/ SQL with an example.
6. What is literal? How many literals are there in PL/ SQL? Give one example for each literal.
7. What do you understand by subprogram?
8. What are the advantages of using stored function or procedure over a PL/ SQL program?
9. Write a PL/ SQL function to take a number as an input and check whether the given number is an odd or even number.
10. Write a procedure to display the detail of a product, where the P_Code in given as an argument.

    Consider the structure of the Product table as:

Column Name	Data Type	Size
P_Code	Varchar2	15
P_Name	Varchar2	15
Qty_On_Hand	Number	8
Unit_Price	Number	10 , 2

11. What is the use of RETURN Clause in a stored function?
12. Where do procedures and functions reside?

Long-Answer Questions

1. What is variable? How do you declare a variable. Give few examples for the same.
2. What do you understand by the PL/SQL package? Explain the features of using this package.
3. What is the difference between package function and package procedure?
4. What is exception handling in PL/ SQL? How many types of exceptions could be handled using exception handling.
5. Explain the system-defined exceptions provided by the Oracle.

NOTES

6. What is transaction? Explain how transactions can be controlled.
7. What is comment? How many types of comments are there in PL/ SQL? Explain with an example.
8. What type of subprograms can you create in PL/ SQL? Discuss.
9. How is data passed in the stored procedure? Explain.
10. How is stored procedure different from stored function?
11. How can an abnormal situation be handled in a subprogram? Explain with example.
12. Discuss the various restrictions on user-defined functions.
13. What is a parameter? How many parameter modes are supported by the sub-programs? Explain.

3.15 FURTHER READING

Snowdon. 1998. *Oracle Programming With Visual Basic.* India: John Wiley & Sons.

Ying Bai. 2021. *Oracle Database Programming with Visual Basic.*NET. India: Wiley-IEEE Press. First Edition.

Byrla. 2017. *Oracle Database 12C*. India: McGraw Hill Education. First Edition.

P.S Deshpande. 2011. *SQL & PL/ SQL for Oracle 11g.* India: Dreamtech Press.

UNIT 4 TRIGGERS, OBJECT RELATIONAL DATABASE, NESTED TABLES AND VARYING ARRAYS

Structure

4.0 INTRODUCTION

A trigger is a PL/SQL code block that automatically triggers (runs) an event. An event in PL/SQL is the data definition language such as INSERT,

UPDATE or DELETE done on a table. Unlike procedure or function, a trigger must be associated with a table. The basic difference between a procedure and a trigger is that a procedure has to be executed explicitly and can have any number of parameters, whereas a trigger is fired (executed) implicitly by the system database.

The goal of object oriented programming develop software which is correct, reliable and maintainable, and which satisfies the requirements of users. Software development is not a static process. It needs to be modified or redesigned according to changes in users' requirements, business rules and strategies. In addition, the complexity of the software also increases. With the development of computers, different approaches to programming have been developed to cope with the dynamic nature and complexity of software. These approaches are known as programming paradigms. A programming paradigm (programming methodology) describes the structure of a program. C++ is a powerful programming language which supports the principles of object oriented programming, such as data abstraction,

encapsulation, inheritance and polymorphism. C++ is particularly suitable for use in the development of reusable software building blocks in the form of class libraries.

In this unit you will study about the introduction of database triggers, types of trigger, object relational database, features and benefits of object-oriented programming, introduction of varying array and nested tables.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the uses and types of database triggers
- Explain the object relational database
- Discuss the concept of object view
- Describe the manipulating data through object view
- Explain the various features and benefits of object-oriented programming
- Discuss the concept of varying array and nested tables

4.2 TRIGGERS AND ITS TYPES

A database trigger helps in maintaining the organization's database in such a manner that without executing the PL/ SQL code explicitlys, updates and validates the data. Triggers can provide a customized management system of your database.

The database trigger can be used to serve the following purposes:

- To enforce integrity constraints (e.g. check the referenced data to maintain referential integrity) across the clients in a distributed database
- To prevent invalid transactions in database
- To update data automatically (one or more tables or views without user interaction)
- To automatically generate derived column values
- To customize complex security authorizations
- To permit insert, update or delete operations to a associated table only during a predetermined date and time
- To provide auditing
- To provide transparent event logging
- To prompt information about various events taken on database, events of users and SQL statements to subscribe applications
- To maintain replication of synchronous table
- To gather statistics on various table accesses

On the basis of different events that occur as before or after triggers are categorized in the following types:

Row Triggers

The row trigger is fired by the system for each and every row affected by the triggering statement. For example, if you are updating salaries of employees on a certain condition and 200 rows are satisfying that condition, the trigger would fire 200 times for each row. If no rows satisfy the condition and no row is updated, the row trigger would not fire even once.

Statement Trigger

Unlike row trigger, a statement trigger is fired only once. Even multiple rows are affected by the triggering statement. For example, if youstill want to delete all the outdated books from your library and you have associated a `DELETE` trigger on Book table on a certain condition. If you delete records from the book table and fifty-five rows satisfy that delete condition, the statement trigger would fire only once. If no rows satisfy the condition and no row is deleted, the statement trigger would not fire even once.

By default, the trigger is a statement trigger.

Before Triggers and After Triggers

When you define a trigger, you could specify the timing for its execution. This timing either could be before or after. The specified timing decides when this trigger would be fired by the system.

Like `row` and `statement` triggers these triggers are also fired by DML statements associated on a table. Remember that, if insert, update or delete command are executed on a view associated to the same table to which your trigger is associated, the triggers would not be fired by the system.

`BEFORE` and `AFTER` apply to both statement and row triggers.

The `BEFORE` trigger is fired by the system before executing the triggering statements.

This trigger runs the trigger action before the triggering statement is run.

For example, a company wants to maintain the records of all the employees irrespective of whether they are still working with the company or not. Data for the employees who are working with the company is maintained in the Employee table and data for the employees who have left the company is maintained in the `Emp_History` table. You can associate a `BEFORE` trigger on an `Employee` table on `DELETE` event and write the trigger action to store the record to be deleted from the Employee table to the `Emp_History` table. You can also use the before trigger to derive the value for a column before inserting or updating a record in the table.

The `AFTER` trigger is fired by the system after exciting the trigger statement. Like `BEFORE` trigger it also applies to both row and statement trigger.

Instead of Triggers

Instead of triggers are those triggers which provide a way to modify views that cannot be modified directly through DML statements `INSERT, UPDATE` and `DELETE`. Unlike other types of triggers Oracle fires the trigger instead of executing the triggering statement.

The simple DML statements, such as `INSERT, DELETE` and `UPDATE` can be written either against the table or view. The `INSTEAD OF` trigger is fired to update the associated tables appropriately. `INSTEAD OF` triggers are activated for each row of the view that gets modified.4.2.5 Triggers on System Events and User Events

Triggers can be used to prompt information about database events to the database users.

These database events can include:

- System events
  - Database startup and shutdown
  - Data Guard role transitions
  - Server error message events
- User events
  - User logon and logoff
  - Data Definition Language (DDL) statements (`CREATE, ALTER` and `DROP`)
  - DML statements (`INSERT, UPDATE` and `DELETE`)

Triggers on system events can be defined at the database level or schema level. The DBMS_AQ package is one example of using database triggers to perform certain actions.

Triggers on system events can be defined at the database level or schema level. The DBMS_AQ package is one example of using database triggers to perform certain actions.

For Example, a database shutdown trigger is defined at the database level:

```
CREATE TRIGGER register_shutdown
ON DATABASE
SHUTDOWN
BEGIN
...
DBMS_AQ.ENQUEUE(...);
...
END;
```

Triggers on DDL statements or logon/logoff events can also be defined at the database level or schema level. Triggers on DML statements can be defined on a table or view. A trigger defined at the database level fires for all database users, and a trigger defined at the schema or table level fires only when the triggering event involves that schema or table.

Combination of Trigger Types

Using the combination of the above types of trigger four types of triggers are possible:

1. **BEFORE Statement Trigger**

The trigger action is run before executing the triggering statement.

2. **BEFORE Row Trigger**

   The trigger action is run before modifying each row affected by the triggering statement and before checking appropriate integrity constraints, if the trigger restriction was not violated.

3. **AFTER Statement Trigger**

   The trigger action is run after executing the triggering statement.

4. **AFTER Row Trigger**

   After modifying each row affected by the triggering statement and possibly applying appropriate integrity constraints, the trigger action is run for the current row provided the trigger restriction was not violated. Unlike `BEFORE` row triggers, `AFTER` row triggers lock rows.

Multiple triggers of the same type for the same statement could be associated on a table. For example, one `BEFORE` statement triggers and one `AFTER` statement triggers for `UPDATE` statements can be associated on the same table at a time.

As many triggers of the preceding different types can be created as you need for each type of `INSERT`, `UPDATE` or `DELETE` statement.

4.2.1 SQL *Forms vs Database Triggers

SQL *Forms—an Oracle tool can also define various triggers as in Oracle SQL *Plus. Trigger can be stored and executed for further use as a part of an application developed using SQL *Forms. There are various differences between database trigger and SQL *Forms triggers.

The difference between database triggers and SQL *Forms triggers are given in Table 4.1

Table 4.1 *Differences between Database Triggers and SQL *Forms*

Database Triggers	SQL *Forms Triggers
These execute or fire when a data manipulation language (DML) operation is performed on its associated table.	These execute or fire when a user navigates between fields on the screen or presses a key at run time.
These can be row level or statement level.	These have no distinction between row level and statement level.
These are executed implicitly by Oracle.	These are executed explicitly by the user.
These can manipulate data stored in Oracle tables via SQL commands.	These can manipulate data in Oracle tables as well as in form variables.
These can be fired from any session executing triggering DML statements.	These can be fired only from the form that defines the trigger.
These can cause other database triggers to fire.	These can cause other database triggers to fire, but not other form triggers.

NOTES

Database Triggers vs Declaration Integrity Constraints

Both declarative integrity constraints and trigger are used to maintain data integrity. Although their function is the same they still differ from each other in various ways.

The difference between database triggers and declaration integrating constraints are as follows:

- A declaration integrity constraint applies to the data that exists in the table and is always true. These include attributed based constraints, tuple based constraints, key and referential integrity constraints. These constraints are checked by the systems for violation of the constraints on actions that may cause a violation such as insertion, updation or deletion of data. If any action causes the violation of the constraints, the system aborts the action accordingly to maintain data integrity.
- A trigger does not apply on stored data; it works during the transaction on its associated table. A trigger is fired automatically by the system when a specified event occurs. The events which may cause a trigger execution are an insert, delete or update command.

Structure of PL/SQL Trigger

Like stored procedures and stored functions, trigger is also a stored procedure. The only difference is that triggers are run automatically by the database whenever some event such as insert, update or delete operations occur.

Like a PL/SQL codes block, procedure and function are also divided into different sections.

The syntax for creating a trigger

```
CREATE [ OR REPLACE ]
TRIGGER <trigger_name>
BEFORE (or AFTER)
INSERT OR UPDATE [OF COLUMNS] OR DELETE
ON table_name
[FOR EACH ROW [WHEN (condition)]]
DECLARE
Declaration statements
…
BEGIN
Executable statements
...
EXCEPTION
Exception handling statement
…
END;
```

A database trigger could also have declarative and exception handling parts.

How to Apply Triggers

A database trigger has three sections which can be applied for various PL/SQL functions. These sections are as follows:

1. Trigger Statement

A trigger statement specifies the data manipulation language (DML) statement such as insert, update or delete that causes a trigger to be executed. In trigger statement, you specify the table to which the trigger would be associated.

The list of various statements, including `INSERT, UPDATE [OF COLUMNS]` or `DELETE` refers to statements that fire this trigger. All these three commands could be specified or just one could be specified as per the business requirement. For example, if you want to fire a trigger when you delete record of a table then you need to specify a `DELETE` in the list. This trigger would fires even if you delete the record from a view which is associated with the same table to which your `DELETE` trigger is associated.

Whereas a `per` statement trigger would fire just once for all statements, even if an `update` statement is updating forty rows.

2. Trigger Body Action

A trigger body is a PL/ SQL code that is executed when the trigger is fired. The PL/SQL block is written between `BEGIN` and `END` statements and is a usual code block where you can place PL/SQL commands. The PL/SQL code may include DML statements, control statements (e.g. `if else`, `End if` and `loop`, …) so on and so forth.

Like other PL/SQL code block, a trigger has a restriction to use Transaction Control Language. The `COMMIT` and `ROLLBACK` commands could not be used within a trigger.

3. Trigger Restriction

One of the trigger is a `row` trigger where the restriction can be included by using `WHERE` clause. The condition in the `WHERE` clause is evaluated for each row which is affected by the trigger.

4.2.2 Create a Trigger

XYZ company has the employee detail in employee table. The company wants to have the history of all the employees who have left the organization. To store the employee history, a new table emp_history is created with the same structure as employee table.

The structure of an employee table is given in Table 4.2

Table 4.2 *Employee Table*

Column Name	Data Type	Size
EMP_CODE	NUMBER	10
E_NAME	Varchar2	15
DESIGNATION	Varchar2	35
SALARY	NUMBER	10,2
DEPTNO	NUMBER	2

NOTES

The employee table contains the records shown in Table 4.3.

Table 4.3 *Employee Table*

EMP_CODE	E_NAME	DESIGNATION	SALARY	DEPTNO
7369	SMITH	CLERK	15000	20
7499	ALLEN	SALESMAN	35000	30
7521	WARD	SALESMAN	32000	30
7566	JONES	MANAGER	55000	20
7654	MARTIN	SALESMAN	30000	30
7698	BLAKE	MANAGER	60000	30
7782	CLARK	MANAGER	64000	10
7788	SCOTT	ANALYST	58000	20
7839	KING	PRESIDENT	70040	10
7844	TURNER	SALESMAN	30430	30
7876	ADAMS	CLERK	23000	20

Create a Duplicate Table

To maintain the employee history a table, `emp_history` can be created with the SQL command as follows:

```
SQL> Create table emp_history as select * from employee
where emp_code is null;
```

The above command would create a new table, `emp_history` which would contain all the fields of employee table (as `*` represents all the fields of a table). The where condition '`emp_code is null`' is used to create the duplicate table empty. Without where the clause duplicate table would contain all the records of employee table.

Table created

You could see the structure of new table `emp_history` by giving the following command:

```
SQL> Desc emp_history;
```

Table 4.4 *Emp_History*

Column Name	Data Type	Size
EMP_CODE	NUMBER	10
E_NAME	Varchar2	15
DESIGNATION	Varchar2	35
SALARY	NUMBER	10,2
DEPTNO	NUMBER	2

When any employee leaves the organization his or her detail would be deleted from the `Employee` table. The same record should be inserted into `emp_history`

table. A trigger can be associated on table employee on the event delete.

The code for the trigger is as follows:

Example : Before DELETE Trigger

```
/* This is a trigger which is associated with the employee
table and would fire on delete command */

Create or replace trigger emp_history
before Delete on employee
for each row

DECLARE

 – Declare the variables.
EMP_CODE NUMBER(10);
E_NAME VARCHAR2(15);
DESIGNATION VARCHAR2(35);
SALARY NUMBER(10,2);
DEPTNO NUMBER(2);

BEGIN
–Copy the data to be deleted from employee table into
variables
EMP_CODE:=:old. emp_code;
E_NAME :=:old.E_NAME;
DESIGNATION:=:old. designation;
SALARY:=:old.salary;
DEPTNO:=:old.deptno;
–insert the delete record into employee history table
insert into emp_history values (emp_code, e_name,
designation , salary, deptno);
end;
/
```

In the above example, `emp_history` is a trigger which is associated with the employee table. This trigger would fire on delete command on `employee` table and would store the deleted record in `emp_history` table.

Naming a Trigger

As duplicate names cannot be used for tables, views procedures, package and function, trigger names must also be unique with respect to other triggers in the same schema. Trigger name can be duplicate in different schema.

Two different objects such as a table and a trigger can have the same name.

Compiling Triggers

To compile the trigger write the following statement at SQL prompt:

```
SQL> @emp_history
```

Triggers are similar to PL/ SQL blocks. They have the additional capabilities to carry :NEW and :OLD data values. Each time a PL/SQL code block is loaded into memory it needs to be compiled. There are three stages of compilation which are as follows:

1. Syntax checking
2. Semantic checking
3. Code generation

In the syntax checking stage, the PL/SQL syntax is checked for correctness. In this stage, a parse tree is generated.

In the semantic checking stage type checking is done. In this, further processing on the parse tree is also done.

In the code generation stage the pcode is generated.

Unlike procedures and functions, triggers are fully compiled when the CREATE TRIGGER statement is entered, and the pcode is stored in the data dictionary.

When a trigger is fired, opening of a shared cursor is not required to execute the trigger action. It helps in executing the trigger directly.

Testing a Trigger

To test whether the trigger is fired and inserted in the deleted record in emp_history table, delete a few records from employee table as shown:

```
SQL> delete from employee where emp_code = 7782;
```

The above command would delete a record from employee table where emp_code is 7782. Now to check whether this record has been inserted into emp_history table or not, write the following command on SQL prompt:

```
SQL> Select *from emp_history;
```

This command would prompt the record as shown in table 4.5.

Table 4.5 *Emp_history*

EMP_CODE	E_NAME	DESIGNATION	SALARY	DEPTNO
7782	CLARK	MANAGER	64000	10
7876	ADAMS	CLERK	23000	20
7844	TURNER	SALESMAN	30430	30

BEFORE INSERT Trigger

In the example, a trigger is associated with the employee table. This trigger would fire before inserting a new record in the table.

```
create or replace trigger insert_emp
before Insert on employee
for each row
begin
DBMS_OUTPUT.PUT_LINE('New employee Code inserted is: '
|| :NEW. emp_code);
```

```
DBMS_OUTPUT.PUT_LINE('New employee Name inserted is :'
|| :NEW.e_name);

end;
/
```

NOTES

In the given example `insert_emp` is a trigger which is associated with the employee table. This is a trigger that would fire on insert command on the `employee` table and would prompt new employee code and employee name before inserting it into the employee table.

Testing

To test whether the trigger is fired and displays message on screen, insert new record in to employee table as shown:

```
SQL> Insert into employee (emp_code, e_name) values
(321,'Scott');
```

When new record is inserted into the employee table the system prompts the message as shown bellow:

New employee `Code` inserted is `:321`

New employee `Name` inserted is `:Scott`

1 row created.

Make sure that serveroutput is on. In case it is not, write the following command on SQL prompt:

```
SQL> set serveroutput on
```

The trigger would execute even if you insert data in all the fields of employee table.

4.2.3 IF Statement in Trigger

To control the PL/SQL code execution `IF` statement is used. Similarly, a database trigger also uses `IF` statement. `IF` statement in database triggers is used to determine what statement caused the execution of the trigger, such as inserting, updating or deleting a data from the associated table.

The general form of `IF` statements in trigger are as follows:

- `If Inserting Then`
- `If Deleting Then`
- `If Updating Then`

An example of `IF` statement in trigger is as follows:

```
create or replace trigger emp_trigger
before insert or update or Delete on employee
for each row
begin
/* the trigger would fire either by inserting, updation
or deleting the record from employee table and the following
conditions would be checked */
```

```
if inserting then
    dbms_output.put_line(' Inserting Employee ' ||
:new.e_name);
elsif deleting then
    dbms_output.put_line(' Deleting Employee ' ||
:old.e_name);
elsif updating then
    dbms_output.put_line(' Updating Employee ' ||
:old.e_name || ' to ' ||:new.e_name);
end if;
end;
/
```

In the given example, `emp_trigger` is a database trigger which is associated with the employee table. This trigger has three IF conditions to determine what statement invoked it, and prompts an appropriate message in various cases.

Different conditions of trigger execution are as follows:

1. Insert record into employee table

To insert a record into the employee table the following command is given:

```
SQL> insert into employee (emp_code, e_name, designation)
values (1001,'xyz','manager');
```

When inserting a record into employee table the first condition is true and the system would prompt a message as shown:

```
Inserting Employee xyz
New employee Number inserted is :1001
New employee Name inserted is :xyz
1 row created.
Deleting Employee KING
```

2. Delete record from employee table

To delete a record from the employee the following command is given:

```
SQL> delete from employee where emp_code=7839;
```

When deleting a record from employee table the second condition is true and the system would prompt a message as shown:

```
Deleting Employee KING
1 row deleted.
```

3. Update record an employee table

To update record on an employee table the following command is given:

```
SQL> update employee set e_name='Spark' where emp_code=7934;
```

When updating a record from employee table the third condition is true and the system would prompt a message as shown:

```
Updating Employee MILLER to Spark
1 row updated.
```

NOTES

Privileges Required for Creating Trigger

Certain privileges are required for creating triggers which are not required to create procedures and functions.

The database administrator (DBA) has the right to grant you the permission to create triggers. Its syntax is as follows:

```
SQL> GRANT CREATE TRIGGER to <user name>
```

For example, the user 'Scott' does not have the right to create a trigger, the administrator would grant the permission by using the following command.

```
SQL> GRANT CREATE TRIGGER TO Scott;
```

4.2.4 Trigger States

Trigger exists in two states. These states are as follows:

1. Enabled State

If a trigger is in an enabled state and triggering statement is entered and the trigger restriction evaluates to TRUE, the trigger executes its trigger body.

2. Disabled State

If a trigger is in a disabled state and triggering statement is entered and the trigger restriction evaluates to `TRUE`, the trigger does not execute its trigger body.

By default, a trigger is created in an enabled state. A `DISABLE` clause of the `CREATE TRIGGER` statement is used to to create a trigger in a disabled state.

Viewing Triggers

To view all the triggers created by the user a data dictionary named `USER_TRIGGERS` can be used.

To see all the triggers use select statement on USER_TRIGGERS as shown:

```
SQL> Select trigger_name from user_triggers;
```

For more description you could also write the following command:

```
SQL> Select * from user_triggers;
```

Dropping (Deleting) a Trigger

You can delete triggers that cease to be essential, or cause unnecessery action on your database. You can delete a trigger by using the DDL command:

DROP

The syntax is as follows:

```
SQL >  Drop trigger < trigger name >
```

Example:

```
SQL>   Drop trigger emp_history ;
```

Check Your Progress

1. What does a database trigger help in maintaining?
2. State the only difference between stored procedure and triggers.
3. List the different types of triggers.
4. Write the statement to compile a trigger at SQL prompt.
5. List the general form of IF statements in trigger.
6. Name the two states in which trigger exists.

4.3 OBJECT RELATIONAL DATABASES

A data set administration framework (DBMS) made up of both a social data set (RDBMS) and an item organised data set is known as an article social information base (ORD) (OODBMS). In its outlines and query language, ORD preserves the key components of any item-based data set model, such as articles, classes, and heritage.

An item social data set administration framework can also be referred to as an article social data base (ORDBMS). Relational information models and object-based information models are both very useful. In any event, it was believed that the two of them were lacking in some qualities, therefore work began on creating a model that combined the two. As a result of the examination that was finished in the 1990s, the Object social information model was created.

Because it comprises viewpoints and attributes from both models, ORD is designed to act as a link between social and article-based data sets. Because the information is stored in a traditional data set and controlled and accessed using questions expressed in an inquiry language like SQL, the basic methodology in ORD is based on RDB. ORD, on the other hand, is a product-oriented trademark in which the data set is considered as an article store, typically for programming done in an item-oriented programming language. APIs are used to store and access the data as items in this case.

One of ORD's main goals is to break down any barriers between theoretical information and techniques for social and article-based data sets, such as the substance relationship graph (ERD) and item social planning (ORM). It also intends to connect the gap between social data sets and item-based demonstration methods, which are commonly used in programming languages such as Java, C#, and C++.

Traditional RDBMS products are concerned with the efficient association of data obtained from a limited set of data types. An ORDBMS, on the other hand, offers a component that allows designers to create and enhance their own data types and methodologies that may be applied to the DBMS. ORDBMS hopes that this will allow developers to spend more time thinking about the problem area.

4.3.1 Features and Benefits of Object Oriented Programming

Initially, when computers were invented, the binary language was used to write

programs. However, as programs grew in size, it became difficult to write programs using binary language. Then the assembly language, though also not user friendly, was developed to write large programs. With changes in users' requirements, the size and the complexity of the programs continued to grow which led to the development of high level languages, such as Beginner's All-Purpose Symbolic Instruction Code or BASIC and FORmula TRANslation or FORTRAN.

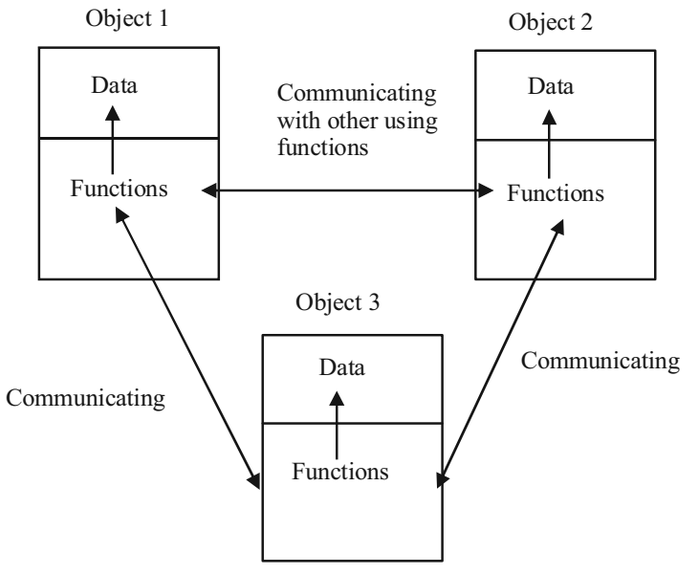
However, these languages provided an unstructured way of writing programs. In the *unstructured programming paradigm*, all the instructions of a program were written one after the other in a single function and, hence, these languages were suitable for writing only small and simple programs. For large and complex programs, it became difficult to trace and debug errors.

To overcome the limitations of unstructured programming paradigm, other programming paradigms, namely procedural and Object Oriented Programming or OOP paradigms were developed which help the programmers to develop the programs in a structured way.

Object Oriented Programming Paradigm

To overcome the limitations of procedural programming, the OOP paradigm has been developed that has revolutionized the process of software development. It not only includes the best features of the structured programming, but has also introduced some new and advanced features that the procedural programming lacked. The most important feature is that unlike the procedural approach in which the program is divided into a number of functions, OOP divides the program into a number of objects. An *object* is a unit of structural and behavioural modularity that contains a set of properties (or data) as well as the associated functions. In addition, programmers can create relationships between one object and another.

The functions of the object (also known as member functions) provide the only way to access the object's data. If a user wants to read or manipulate any data item, then it is possible only if the member function to do the same is available in the object. Therefore, the data is hidden from the outside world, and hence safe from accidental modifications. The basic idea behind OOP is shown in Figure 4.1.



NOTES

Fig. 4.1 *Data and Functions in OOP*

As stated earlier, OOP has introduced some new and advanced features that the procedural programming lacked. The most important feature is that unlike procedural approach in which the program is divided into a number of functions, OOP divides the program into a number of objects. Some of the other features of OOP are also described here.

- OOP emphasises on data rather than the functions or the procedures.
- OOP models the real world very well by binding the data and associated functions together under a single unit and thus, prevents the free movement of data from one function to another.
- The data of one object can be accessed by the associated functions of that object only. Other functions are not allowed to access that data. In other words, data is hidden from the outside world. However, the functions of one object can access the functions of other object.
- The objects of the entire system can interact with each other by sending messages to each other.
- The programs written in OOP are easy to maintain and extend as new objects can be easily added to the existing system whenever required without modifying the other objects.
- OOP follows the bottom-up approach for designing the programs. That is, first objects are designed and then these objects are combined to form the entire program.

Approach of OOP Principles

C++ is a powerful programming language which supports the principles of object oriented programming, such as data abstraction, encapsulation, inheritance and polymorphism. C++ is particularly suitable for use in the development of reusable software building blocks in the form of class libraries. This high level language permits selective use of the advantages of object oriented programming. C++ supports the creation of class libraries. Class libraries are reusable software building blocks.

C++ avoids runtime errors by strict type checking. This greatly improves the stability of the programs. C++ supports object oriented programming which is based on the following principles:

- **Data Abstraction:** The data abstraction refers to the act of representing the essential features without including the background details or explanations. This concept uses the classes and objects. Classes use the concept of abstraction and are defined as a list of abstract attributes, such as size, weight and cast functions to operate on these attributes. The attributes are sometimes called data members because they hold information. The reasons of abstraction are defined as follows:

- **Flexibility in Approach:** By hiding data or abstracting details that are not needed for presentation, the programmer achieves greater flexibility in approach.
- **Enhanced Security:** Abstraction gives access to data or details that are needed by users and hide the implementation details, giving enhanced security to application.
- **Easier Replacement:** With the concept of abstraction in object oriented programming language, it is possible to replace code without recompilation. This makes the process easier and saves time for users.
- **Modular Approach:** In object oriented programming language C++, the abstraction concept helps users to divide the project application into modules and test each of them separately. Then all modules are integrated and ultimately tested together. This approach makes the application development easier.

There are various ways of achieving abstraction in object oriented programming language C++. One approach is to take modular based code that is broken apart into smaller segments known as functions. This functional or modular approach helps the code to be reused again and again when needed. For example, a programmer might write a function for computing an average and another programmer might write a function for computing salary. These functions can be reused when needed by anyone. The modular based approach helps to centralize all data of a similar type, under the control of a type module. Defining module types allow the module to be an abstract data type. Data abstraction lets you to create new data types which are not available in the programming languages. It is, in fact, is a process of representing the essential features without including implementation details. It is the process of defining function and is used several times as needed with the required properties. With reference to object oriented programming, `class student` is designed to represent the data elementary. These data elements can be `student ID, name, marks, grade`, etc., and in the same way, for `class car`, `car model name, colour, price`, etc., are considered as data elements. The classes designed to object oriented language is also known as user defined data types. In programming language, standard or built in data type has two attributes. These attributes refer to a set of values and a collection of allowable operations for the defined values. The instances are commonly known as variables. Often, the high level languages are not primitive to correlate with the real world data, for example `Password, StudentList` and `BusSchedule`, etc., data types are not supported by C++. Data abstraction can be implemented by two methods. In first method, abstraction can be implemented if data representation is chosen for the abstract data. It is noted that data types already exist in the chosen programming language. The second method implements each allowable operation in terms of program instruction. The four terms used in data abstraction are as follows:

- **Object Instance:** Each single entity in an object model is called an object instance. For example, the Figure 4.2 represents a single object instance, i.e., Car.

Fig. 4.2 *Single Entity of Class Car*

The object model directly supports references of the objects. As in reality, most object instances 'reference' each other in some way. References are also called 'relationships'.

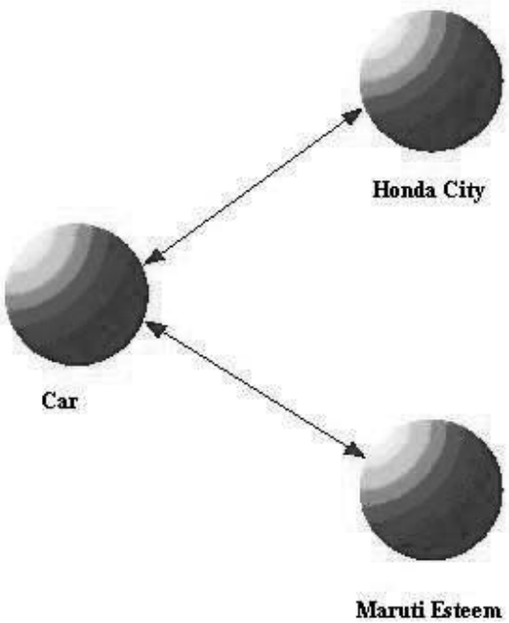


Fig. 4.3 Object Identifications of Object Car

Figure 4.3 shows two references. The object instance representing Car stores two OIDs. One OID represents **Honda City**. The other OID represents **Maruti Esteem**. The OIDs appear as arrows. These references represent the parents of Car**.** The objects representing **Honda City** and **Maruti Esteem** store an OID that represents Car**.**

- **Object Identity:** Object identity in object models means that every object instance has a unique, unchanging identity. Object identification is often referred to as an OID. OIDs are used to refer the object instances. They are independent of data contained in the object. The internal data values are not used to generate identification and also are generated by the object system. Users or programs have no control over identification

for object identity. They last the lifetime of the object. The OID never changes even when the data contents may change.

- **Class and Objects:** Object instances are grouped together into a class. The class defines the structure and the attributes or fields for each of the objects. In this example, there is a Car class that has possible attributes of 'car model' and 'price'. It also has relationships of parents and children.

- **Encapsulation:** Objects encapsulate states and functions. In C++, objects are described by means of class definitions. A class definition collectively defines data and the functions that operate on this data. Software produced according to this principle is more robust, easier to maintain and easier to extend since there are fewer dependencies between the modules and the details of the implementation are encapsulated in classes.
- **Inheritance:** Classes can inherit attributes from other classes. Inheritance permits better structuring of the software and helps in reducing the amount of code, as common sections of code can be reused.
- **Polymorphism:** Objects of different types can share a common function interface, enabling a developer to use the various objects without needing to know their type. The use of polymorphism produces software that is more general purpose, more flexible and more reusable.

Basic Concepts of OOP

To understand the concept of object oriented programming, it is necessary to know the fundamental terms and concepts of this approach. These include objects, classes, data abstraction, encapsulation, inheritance, polymorphism and message passing.

Objects

Objects are the small, self-contained and modular units with a well defined boundary. An object consists of a state and behaviour. The *state* of an object is one of the possible conditions that an object can exist in and is represented by its characteristics or attributes or data. The *behaviour* of an object determines how an object acts or behaves and is represented by the operations that it can perform. In OOP, the attributes of an object are represented by variables and the operations are represented by functions.

An object biscuit, for example may consist of data product code P001, product name Britannia biscuits, price 20 and quantity in hand 50. These data values specify the attributes or features of the object. Similarly, consider another object noddles with product code P002, product name Maggi Noodles, price 10, and quantity in hand 20 (refer Figure 4.4). In addition, the data in the object can be used by the functions, such as `check_qty()` and `display_product()`. These functions specify the actions that can be performed on data.

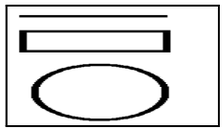
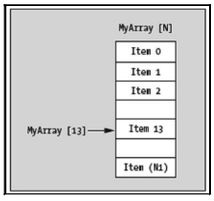
Objects are what actually runs in the computer and, thus, are the basic runtime entities in object oriented systems. They are the building blocks of object oriented programming. Although, two or more objects can have the same attributes, still they are separate and independent objects with their own

identity. In other words, all the objects in a system take a separate space in the memory independent of each other. Note that the main objective of breaking down complex software projects into objects is that changes made to one part of a software should not adversely affect the other parts.

Some Real Life Examples of Objects

When you consider a programming problem in an object oriented language, you need to determine how the problem will be divided into functions, and objects and class. With the help of class, objects and functions user friendly programs can be designed using object oriented language which are known as 'real life programs'. This approach is helpful in finding the close match between objects in the programming sense and objects in real world. Table 1.1 summarizes the various types of objects used in real life as well as programming environment.

Table 4.6 *Objects used in Real Life and Programming Environment*

Types of Objects	Examples	Picture
Real Life Objects	Examples of real life objects are electrical components in a circuit design program, aircraft in an air traffic control system, countries in an economics model, etc.	
Graphic Objects	Examples of graphic objects are lines, rectangles, circles, etc.	
Computer User Environment Objects	Examples of computer user environment objects are Windows, menus, mouse, keyboard, etc.	
Computer Game Objects	Examples of computer game objects are positions in board game (chess), flowers in an ecological simulation, opponents and friends in adventure games, ghosts in a maze game, etc.	
Programming Objects	Examples of programming objects are customized arrays, stacks, linked lists, binary trees, etc.	

The match between programming objects and real world objects produce a good result of combining data and functions and the resulting objects offer a revolution in program design.

Classes

A class is defined as a user defined data type that contains the entire set of similar data and the functions that the objects possess. In other words, a class in OOP represents a group of similar objects. As stated earlier, in the real world millions of

objects exist and each of them has its own identity. However, each of them can be categorized under different groups depending on the common properties they possess and the functions they perform. Cars, scooters, motorbikes, buses, etc., for example can be grouped under the category vehicles. Similarly, dogs, cats, horses, etc., can be grouped under the category animals. Thus, vehicles and animals can be considered as classes.

A class serves as a blueprint or template for its objects, that is, once a class has been defined, any number of objects belonging to that class can be created. The objects of a class are also known as the *instances* or the variables of that class and the process of creating objects from a class is known as *instantiation*. Note that a class does not represent an object, rather it represents the data and functions that an object will have.

A class `Product`, for example, consists of data, such as `p_code`, `p_name`, `p_price` and `qty_in_hand` that specify the attributes or features of the objects of the Product class. In addition, it consists of functions, such as `display_product()` and `check_qty()` that specify the actions that can be performed on data (refer Figure 1.4).

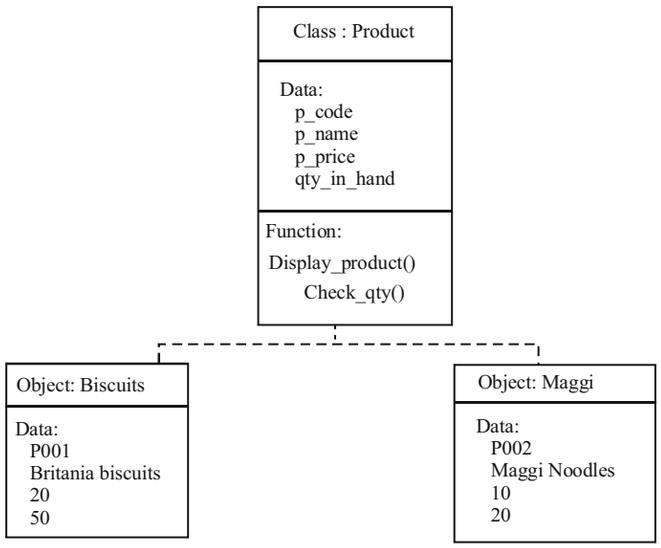


Fig. 1.4 *Class and Objects*

Note that the data belonging to a particular class is known as its *data members* and the functions of the class are known as the *member functions* and both collectively are known as the *members* of the class.

Abstraction

Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time. It allows managing complex systems by concentrating on the essential features only. While driving a car, for example a driver only knows the essential features to drive a car, such as how to use clutch, brake, accelerator, gears, steering, etc., and is least bothers about the internal details of the car, such as motor, engine, wiring, etc.

Abstraction can be of two types, namely (i) Data abstraction and (ii) Control abstraction. *Data abstraction* (also known as data hiding) means hiding the details about the data and *control abstraction* means hiding the implementation details. In object oriented approach, one can abstract both data and functions. However, generally, the classes in OOP are defined in such a way that the data is hidden from the outside world and the functions form the public interface. Thus, the functions of the class can be directly accessed by other functions outside the class, and the hidden data can be accessed indirectly with the help of these functions.

Note that the values of the hidden data members cannot be passed to the outside world unless the functions are written to pass that information outside the class. Since the internal details of the class are hidden from the outside world, data abstraction ensures security of data by preventing it from accidental changes or manipulations by other parts of the program.

Note: Classes in the object oriented programming are also known as Abstract Data Types (ADT) as they use the concept of abstraction.

Encapsulation

Encapsulation is the technique of binding or keeping the data and functions (that operate on them) together in a single unit called a class. Encapsulation is the way to implement data abstraction. A well encapsulated object acts as a 'black box' for other parts of the program, that is, it provides services to the external functions or other objects that interact with it. However, these external functions or the objects do not need to know its internal details. For example, in Figure 1.4 the data `p_code`, `p_name`, `p_price` and `qty_in_hand` and the functions `display_product()` and `check_qty` are encapsulated in a class `Product`.

Inheritance

Inheritance can be defined as the process whereby an object of a class acquires characteristics from the object of another class. As stated earlier, all the objects of a similar kind are grouped together to form a class. However, sometimes a situation arises when different objects cannot be combined together under a single group as they share only some common characteristics. In this situation, the classes are defined in such a way that the common features are combined to form a generalized class and the specific features are combined to form a specialized class. The specialized class is defined in such a way that in addition to the individual characteristics and functions, it also inherits all the properties and the functions of its generalized class.

In the real world, for example, all the vehicles cannot be automobiles—some of them are pulled-vehicles also. Thus, car and scooter both are vehicles that come under the category of automobiles. Similarly, rickshaw and bicycle are vehicles that come under the category of pulled-vehicles. Thus, automobiles and pulled-vehicles inherit the common properties of the vehicle class and also have some other properties that are not common and differentiate them. Thus, the vehicles class is the generalization of automobiles and pulled-vehicles class and automobiles and pulled-vehicles classes are the specialized versions of the

vehicles class. Note that while inheriting the vehicle class, the automobiles and pulled-vehicles do not modify the properties of the vehicle class, however, they can add new properties that are exclusive for them (refer Figure 4.5).

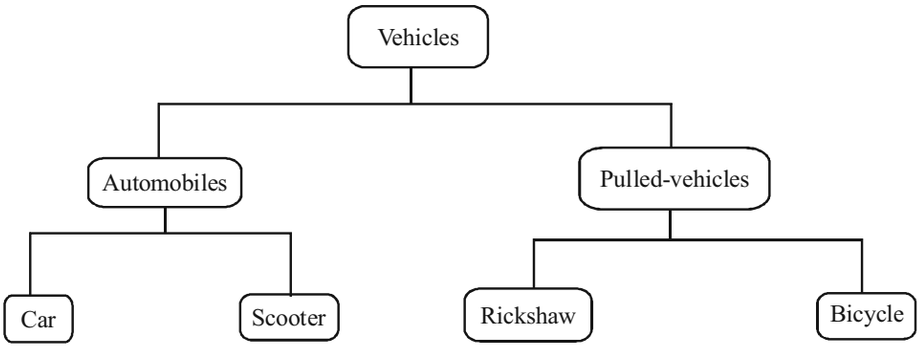


Fig. 4.5 *Inheritanc***e**

In the same way, OOP allows one class to inherit the properties of another class or classes. The class, which is inherited by the other classes, is known as *superclass* or *base class* or *parent class* and the class, which inherits the properties of the base class, is called *sub class* or *derived class* or *child class*. The sub class can further be inherited to form other derived classes. In Figure 1.5, for example, car and scooter are the derived classes of automobiles, and rickshaw and bicycle are the derived classes of pulled-vehicles.

Inheritance can be of two types: (i) Single inheritance and (ii) Multiple inheritance. If a class acquires properties from a single class, it is termed as *single inheritance* and if it acquires characteristics from two or more classes, it is known as *multiple inheritance*. The main advantage of inheritance is reusability. The existing classes can be simply re-used in new software instead of writing a new code. Moreover, new features can be added without altering or modifying the features of the existing class.

Reusability and Extensibility

Inheritance allows code reusability, that is, it facilitates classes to reuse the existing code. It is useful when several classes having similar features are to be created. In such a case, one class is created having common features of all the classes which is used as the base class. Whenever a new class is to be generated, it inherits this base class and only the unique features of the new class are added, thereby avoiding repetition of code. The new class acquires the members of the old class that are already tested and debugged.

The base classes having common features can also be stored in a reservoir so that they can be used by any programmer. These classes stored in the reservoir form part of general-purpose programming tools and new classes generated on the basis of these classes become their specialized versions. Hence, inheritance allows extending and reusing already existing classes, thereby, saving time as well as increasing the reliability. A common class `employee`, for example can be created having some basic features, which can be used by any program requiring classes (such as, `clerk`, `manager`, `part_time_employee`, `full-`

NOTES

`time_employee`, etc.) to be generated having similar features. These features of inheritance play an important role in the program development.

Abstract Classes and Concrete Classes

While inheriting a base class, a derived class not only inherits the data and functions of its base class, but can also provide a different implementation (definition) for the functions of the base class. In such a case, the base class may or may not provide an implementation for its function. It only provides the interface for the functions.

A class that provides only the interface of one or more functions and not their implementations is known as an *abstract class*. An abstract class only specifies what the function does, what all it requires, etc., but it does not specify how the function works. Implementations of such functions are provided in the classes that inherit the abstract class. Note that the instances (objects) of an abstract class cannot be created. This is due to the fact that it does not provide the implementation of the functions. The class that provides an implementation for all its functions is known as a *concrete class*. The concrete classes can have one or more objects. Remember that derived classes that provide implementation of all the functions that have not been implemented in the abstract class are also considered as concrete classes.

Polymorphism and Overloading

Polymorphism (a Greek word meaning *having multiple forms*) is the ability of an entity, such as a function or a message to be processed in more than one form. It can also be defined as the property of an object belonging to the same or different class to respond to the same message or function in a different way. If a message `change_gear`, for example is passed to all the vehicles then the automobiles will respond to the message appropriately, however, the pulled vehicles will not respond. The concept of polymorphism plays an important role in OOP as it allows an entity to be represented in various forms.

In C++, polymorphism can be achieved either at compile-time or at runtime. At compile time, polymorphism is implemented using operator overloading and function overloading. However, at runtime, it is implemented using virtual functions.

Operator overloading is the process that enables an operator to exhibit different behaviour, depending on the data provided, for example, when the '+' operator is used with two numbers, it adds the two numbers and produces the sum. However, if it is used with two strings, it concatenates the two strings and produces the third concatenated string (refer Figure 4.6).

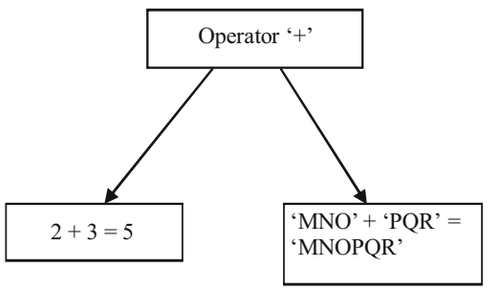


Fig. 4.6 *Operator Overloading*

Similarly, a single function can behave differently depending on the type of data provided. In Figure 4.7, for example, the function `Add()` can be used to add two integers and two float-point numbers. This form of polymorphism is known as *function overloading*.

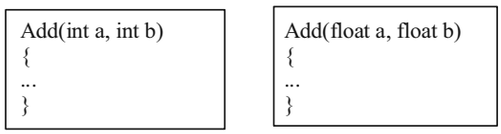


***Fig. 4.7** Function Overloading*

Note: Compile-time polymorphism is also known as static binding, as the linking of function call to the actual code of the function is done at compile-time itself.

Consider another example in which three different classes, `square`, `rectangle and circle are derived from the base class geometrical_shapes`. The function `area()` of the base class is implemented in different ways in all its derived classes, and a call to a particular function is determined at runtime. This form of polymorphism is called runtime polymorphism (refer Figure 4.8).

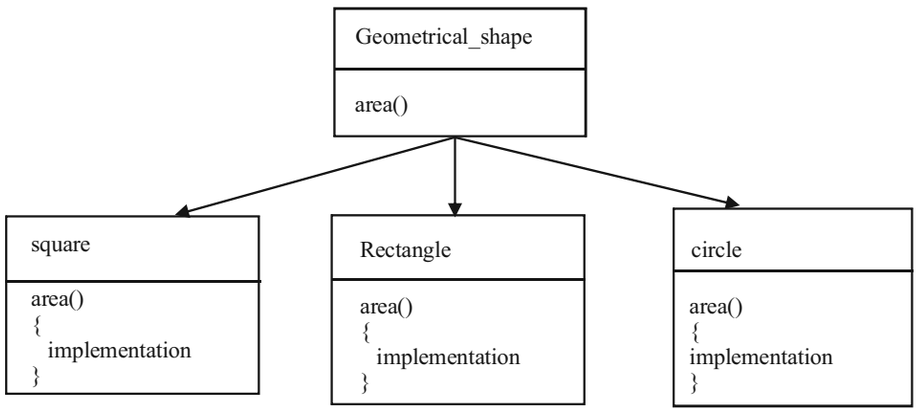


***Fig. 4.8** Runtime Polymorphism*

Message Passing

Message passing is a process of interaction between different objects in a program. As explained earlier, a program following the object oriented paradigm comprises a set of objects each with a set of data and functions. When the program is executed, these objects interact or communicate with each other by sending and receiving messages. The messages are exchanged by calling the member functions of the classes.

Any object of a class that wants to communicate with the object of another class requests the object to invoke the required member function of its class. This function call is different from the normal function call as, in this case, the sending object is sending a request for the execution of the function. However, the receiving object may or may not accept the request depending on whether the function forms the public interface or it is hidden from the outside world. Thus, this form of communication is called message sending and not an ordinary function call.

NOTES

Consider, for example two classes `Product` and `Order`. The object of the `Product` class can communicate with the object of the `Order` class by sending a request for placing order (refer Figure 4.9).

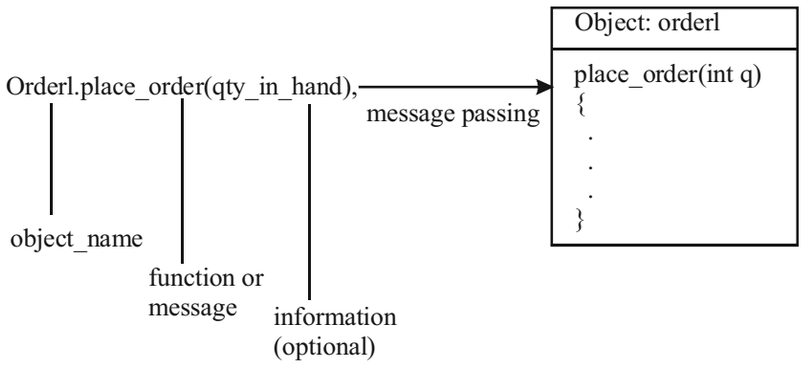


Fig. 4.9 *Message Passing*

Dynamic Binding

Dynamic binding is the process of linking of a function call to the actual code of the function at runtime; that is, in dynamic binding, the actual code to be executed is not known to the compiler until runtime.

The concept of dynamic binding is implemented with the help of inheritance and runtime polymorphism (virtual functions). Consider, for example, the class hierarchy shown in Figure 1.8. Each derived class has the same function `area()`, however, with different body. At runtime, depending on the object being referenced, the desired function will be called.

Benefits of OOP

The object oriented programming paradigm came into use as it overcame certain limitations of the structured and unstructured programming paradigms. The new and advanced features of OOP, such as encapsulation, abstraction, inheritance and polymorphism helped in the development of high quality software. High quality software can be developed due to its certain advantages. Some of the benefits of OOP are as follows:

- In OOP, writing programs with the help of objects is much similar to working with real world objects; that is, the real world objects can be conveniently represented in a program that reduces the complexity of the program and also makes the program structure clear.
- In it, each object is an independent and separate entity that makes modifying, locating and fixing problems in a program an easy task. In addition, any change made inside the class does not affect the other parts of the program. Thus, OOPs are easy to write and easy to maintain.
- In it, data integrity and data security is high, as it focusses on the data and its protection from manipulation by different parts of the program. As a result, OOPs are less error-prone, more reliable and secure.

- Object oriented programs are easy to extend, as new features in a program can be added easily by introducing a few new objects without modifying the existing ones.
- It allows reusability of code, that is, the objects created in one program can be re used in other programs. In addition, new classes can be created with the help of existing ones using inheritance. It leads to faster software development and high quality programs.
- These are easier to adapt and scale, that is, large system can be created by assembling reusable subsystems.

4.4 INTRODUCTION TO OBJECT VIEW

Just as a view is a virtual table, an object view is a virtual object table. Oracle provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables from data of either built-in or user-defined types—stored in the columns of relational or object tables in the database. Object views provide the ability to offer specialized or restricted access to the data and objects in a database. For example, you can use an object view to provide a version of an employee object table that does not have attributes containing sensitive data and does not have a deletion method. Object views allow the use of relational data in object-oriented applications. They let users:

- Try object-oriented programming techniques without converting existing tables.
- Convert data gradually and transparently from relational tables to object-relational tables.
- Use legacy RDBMS data with existing object-oriented applications.

4.4.1 Manipulating Data through Object View

A view object is a class that lets you characterise and deal with a group of columns, usually with the use of a user interface. A view object usually contains a SQL question that selects data from a data set. It tends to be associated with core element elements, allowing you to change data in the data collection, or it may not be associated with element objects at all.

View objects are based on what the client needs to display

A view object uses a SQL query to identify distinct portions of business data that can be identified using credits from substance objects. You create views based on what the consumer requires. The views on information that can be based on, on the other hand, are independent of the fundamental material objects, allowing for flexible information recovery to assist with the required UI. At the end of the day, you can query a large amount of data precisely as you require it to be displayed in the showcase. The view object describes the features of the view column class, which refers to a line in the query result and can also refer to basic substance objects. Customers can look at and change line sets using view objects without having to worry about missing information on the core substance objects. Customers

have control over information by exploring the outcome set, obtaining and establishing characteristic qualities; when the exchange is submitted, modifications are made to the information in the basic data set. View joins are used to communicate connections between view objects. Each view object comes with a built-in iterator that you can use to navigate through the data.

The link between a view object, element object, and the hidden data set table, for example, is shown in the accompanying diagram 4.10. EmpNames is a view object that works on the Emp element object to provide insight into the EMPNO and ENAME portions of the EMP table.

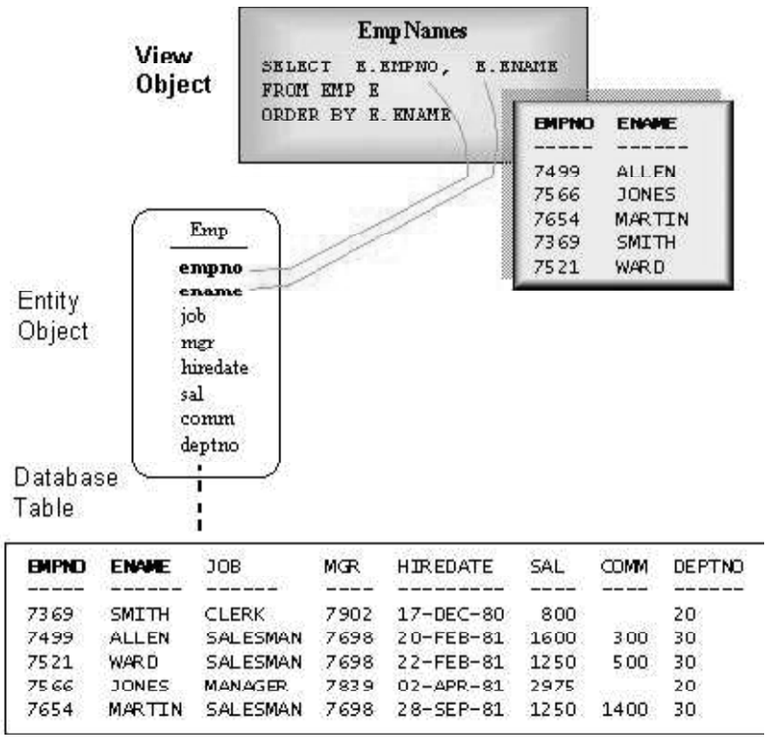


Fig: 4.10 Accompanying Diagram for view object, element object, and the hidden data set

Objects with a view are frequently used to:

- Add an extra layer of security by restricting access to a predetermined set of lines and sections. For example, you may create a view object that excludes segments containing sensitive data, (such as pay rates).
- Keep the complexity of information hidden. A view item, for example, can display parts or lines from multiple substance objects. The fact that the data comes from a few tables is hidden behind such a view object.
- A demonstration tailored to your needs. You can rename segments using a view object without affecting the substance objects on which the view object is based.
- Store complex questions. A query could use table data to do broad computations. The computations are conducted only when the view item's inquiry is executed by saving this query in a view object. Before the information is extracted from the data set, the estimation is run.

- Improve your application skills by using complex SQL that executes quickly and allows you to select only the information you need.

Types of view objects

You can characterize the accompanying general sorts of view objects, either at configuration time or progressively at runtime:

Does the view object contain a SQL query?	Does at least one view attribute map to an entity attribute?	You can use the view object for...	And you get these benefits...
yes	yes	querying, updating, inserting, and deleting data	data consistency with other view objects that map to the same entity objects, and memory savings
yes	no	querying data	less overhead (and potentially faster) than when you map to entity objects, especially when using forward only mode (where there is no view caching)
no	yes	inserting data	no initial query is executed, which saves startup time
no	no	temporary collections of data	can use the same consistent interface for data and easily bind data to a user interface

Furthermore, a datasource other than a data set, such as a level record, accounting page, or XML document, can be used. In this case, you'll need to provide custom code that allows a view object to browse data from the datasource while element objects create data for the datasource.

A view object can map to multiple entity objects

Different view objects can be described for each element object, and a view article can select information from many element objects. Changes made by one view object are rapidly accessible to other view objects in a comparable exchange because information is saved at the substance object level, and all view object references inside a similar exchange share the reserve.

The adjacent figure 4.11, for example, depicts a view object selecting from various substance objects. DeptEmp and EmpNames are view objects that choose data from the Emp element object. DeptEmp selects data from the Dept substance object as well.

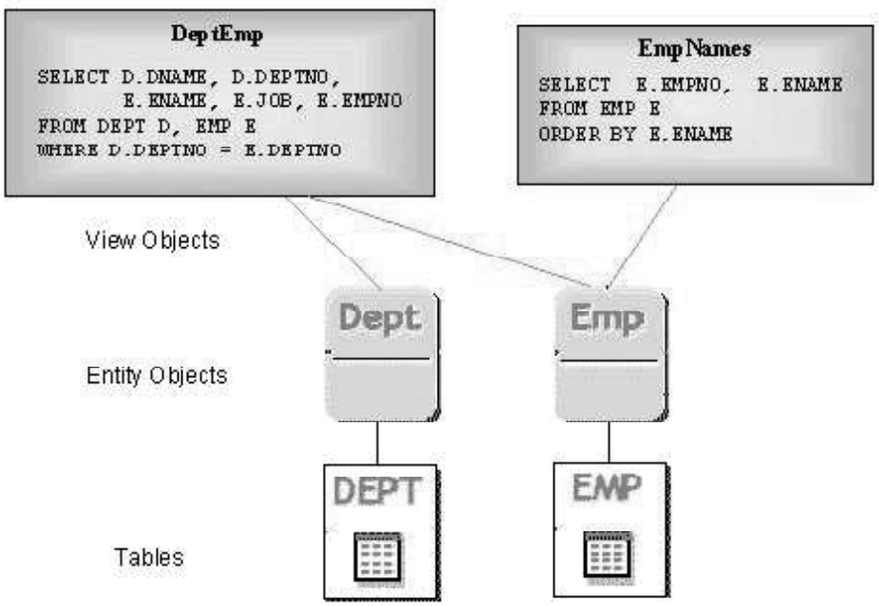


Fig: 4.11 *View object selecting from various substance objects.*

When you characterise a view object, you can figure out which basic substance objects are read only and which backing read/compose technique is employed based on the quality rundown of the view item. When describing a view object based on the EMP and DEPT tables (together with accompanying Emp and Dept element objects), for example, you might specify that Emp data should be editable and Dept data should be introduced as supporting data.

View objects and view links can be part of an application module's data model

You can choose the view connects you need to use to interface view objects in the detail connections in the Application Module Wizard and Editor. View connections can also be represented in code that runs in the background. As a result, detail view objects are synchronised with their respective expert view objects.

A similar view object, such as an unlimited view and a detail view, can be used by an application module at least once or twice; pseudonyms can be used to address the different uses of a view object.

An application module using an expert detail view (DeptView) and a high level view, for example, is shown in the following diagram (EmpView). A view interface might provide viewpoints on the DEPT and EMP tables, as well as a high-level perspective on the EMP table. The DeptView goes by the moniker MyDeptView. MyEmpDetailView (used in an expert detail connection) and MyEmpView are two names for the view object EmpView (used to introduce a high level view).

4.4.2 Introduction to Methods

Object techniques, often known as subprograms, are capabilities or strategies that you can specify in an item type specification to carry out the behaviour that

you require of that type of object. To summon the conduct, an application invokes the subprograms.

Subprograms can be developed in any programming language, including PL/SQL. The data set contains techniques developed in PL/SQL or Java. Techniques written in several dialects, like as C, are archived remotely.

Member Methods

Part techniques grant an application access to the details of an item occurrence. For any activity that you want an object of that type to be able to perform, you define a component strategy in the article type. MEMBER FUNCTION or MEMBER PROCEDURE are the terms used to describe non-correlation part approaches. As seen in "Part Methods for Comparing Objects," correlation strategies employ the MAP MEMBER FUNCTION or ORDER MEMBER FUNCTION.

You can declare a capacity get sum() that aggregates the total cost of a buy request's information as an example of a component strategy. This capability for buy request po is called in the accompanying line of code, which returns the sum to sum line items.

dot notation identifies the current item and the technique it calls. sum line items:= po.get sum(); No matter if there are no limits, enclosures are essential.

This part contains these points:

- SELF Parameters in Member Methods
- Part Methods for Comparing Objects

SELF Parameters in Member Methods

Part approaches feature a built-in boundary called SELF that denotes the item event that is currently invoking the strategy.

Although it is possible to pronounce SELF without hesitation, it is not needed. Without the SELF qualifier, it is easier to write component strategies that refer to the traits and approaches for SELF verifiably.

Member Methods for Comparing Objects

Determine a cause for contrasting characteristics of an article type before analysing and organising them. The advantages of a scalar information type, such as CHAR or REAL, are that they have a predetermined request that allows them to be considered. However, there is no established hub of association for an article type, such as a person type, which can have several features of various information types. You can choose between describing a guide strategy or a request approach for looking at things, but not both.

A guiding technique converts object return esteems to scalar qualities and can sort them according to where they are on the scalar pivot. A request method considers the features of two specified articles in a straightforward manner.

Map Methods

Map strategies return values that can be used to compare and arrange data. Any Oracle intrinsic information kinds (apart from LOBs and BFILEs) and ANSI SQL

NOTES

types like CHARACTER or REAL can be used as return esteems. In Oracle Database SQL Quick Reference, look for the specific sections.

Map techniques, for the most part, do assessments on the item's characteristics in order to re-establish esteem.

Examining obj 1 > obj 2 and correlations inferred by the DISTINCT, GROUP BY, UNION, and ORDER BY provisos, which necessitate organising by columns, map techniques are naturally named.

The correlation: obj 1 and obj 2 are two article elements that can be measured using a guide strategy map().

obj_1 > obj_2

is identical to:

obj_1.map() > obj_2.map()

Examinations are comparable for other social administrators.

A subtype can proclaim a guide technique provided that its root supertype announces one.

See "Equivalent and Not Equal Comparisons" for the utilization of guide techniques when contrasting assortments that contain object types.

Order Methods

Request techniques establish direct item linkages. They can't decide the demand for diverse items, unlike plan tactics. They fundamentally inform you that, based on the model used, the current thing is not quite, equivalent to, or more prominent than the item being contrasted with.

A request technique is a capacity for an item (SELF) with one declared boundary that is a similar object. Either a negative, zero, or positive number should be returned by the approach. This value denotes that the item (the unspoken SELF boundary) is not identical to, or more notable than, the proclaimed boundary object.

Similarly to map approaches, if a request strategy is defined, it is naturally invoked whenever two objects of the same type should be examined.

Guidelines for Comparison Methods

You can declare either a guide or a request strategy, but not both. Using SQL proclamations and PL/SQL procedural explanations, you can think about objects in any strategy type. However, assuming you don't publish one of these methodologies, you can just examine objects in SQL statements for uniformity or imbalance. Two objects of the same type are considered identical if the upsides of their comparing ascribe are equal.

When organising or consolidating a large number of articles, use a guiding technique that maps all of the things into scalars and then sorts the scalars. Because it must be called multiple times, a request strategy is less effective (it can look at just two articles all at once). See "Object Comparisons Execution."

Comparison Methods in Type Hierarchies

In a kind progressive system, assuming the root type (supertype) doesn't indicate a guide or a request strategy, neither can the subtypes.

- Map Method in a Type Hierarchy

Assuming the root type indicates a guide technique, any of its subtypes can abrogate it. Assuming the root type doesn't indicate a guide technique, no subtype can determine one by the same token.

• Request Method in a Type Hierarchy

Just the root type can characterize a request technique. In the event that the root type doesn't characterize one, its subtypes can't add one.

Static Methods

Static approaches are invoked based on the article type rather than the occasion. For tasks that are global to the type and do not require referencing the information of a single item example, you use a static technique. There is no SELF boundary in a static strategy.

STATIC FUNCTION or STATIC PROCEDURE are used to declare static strategies.

You invoke a static strategy by using spot documentation to qualify the technique call with the article type's name, for example:

type_name.method()

See "Static Methods" for data on plan contemplations.

Constructor Methods

A function Object() { [native code] } strategy is a capability that profits another example of the client-defined type and establishes the credits' upsides. Constructor approaches are divided into two categories: framework-specific and client-specific.

The catch NEW can be used to summon a function Object(), but it isn't required.

System-Defined Constructors

The framework, of course, defines a function Object() work for all item types with credits. Occasionally, this function Object() is referred to as the characteristic worth function Object().

The name of the function Object() technique for the person typ object type described in Example is the name of the item type, as shown in the accompanying conjuring:

person_typ (1, 'John Smith', '1-650-555-0135'),

User-Defined Constructors

You can also create and instate client-defined types by defining your own function Object() parts. Although the default framework characterised constructors (or property estimation constructors) are convenient to use since they already exist, client characterised constructors have some major advantages in terms of type progress. "Benefits of User-Defined Constructors" is a good place to start. For more on client-defined constructors for assortments, see "Constructor Methods for Collections."

Literal Invocation of a Constructor Method

A call to the function Object() technique with literals (instead of tie factors) or more stringent summons of function Object() techniques is an exacting conjuring of a function Object() strategy.

Collections (Nested Tables & Varying Arrays):

A composite information type is used to hold esteems with several elements. Subprograms can pass entire composite factors as borders, and inward sections of composite factors can be accessed independently. Scalar or composite inward components are both possible. Scalar parts can be used anywhere that scalar factors can be used. Composite parts can be used anywhere that composite factors of a similar type can be used.

The inside pieces of an assortment are referred to as components since they all have the same information type. With this language structure, you can go to every component of an assortment variable by its remarkable list: variable name (index). You can either characterise an assortment type and then create a variable of that kind, or you can use percent TYPE to create an assortment variable.

The interior components of a record, known as fields, might include a variety of information kinds. With this sentence construction, you may get to each field of a record variable by its name: variable name.field name. You can create a record variable by either defining a RECORD type and then creating a variable of that type, or by using percent ROWTYPE or percent TYPE.

Collection Types:

PL/SQL has three assortment types—acquainted cluster, VARRAY (variable-size exhibit), and settled table.

Collection Type	Number of Elements	Index Type	Dense or Sparse	Uninitialized Status	Where Defined	Can Be ADT Attribute Data Type
Associative array (or index-by table)	Unspecified	String or PLS_INTEGER	Either	Empty	In PL/SQL block or package	No
VARRAY (variable-size array)	Specified	Integer	Always dense	Null	In PL/SQL block or package or at schema level	Only if defined at schema level
Nested table	Unspecified	Integer	Starts dense, can become sparse	Null	In PL/SQL block or package or at schema level	Only if defined at schema level

Number of Elements

It is the highest number of components in the assortment, assuming the quantity of components is known. If the quantity of components is unknown, the farthest reaches of the file type will have the most components in the assortment.

Is it more dense or sparse?

There are no gaps between components in a thick assortment; each component between the first and last is described and valued (the worth can be NULL except if the component has a NOT NULL imperative). There are gaps between the components in a small collection.

Status: Uninitialized

There is an unfilled assortment, however it is devoid of components. Invoke the EXTEND approach to add components to an unfilled assortment (depicted in "Broaden Collection Method").

There is no such thing as an invalid assortment (also known as a molecularly invalid assortment). You must instate an invalid assortment to make it a current assortment, either by making it unoccupied or by assigning a non-NULL value to it (for subtleties, see "Assortment Constructors" and "Allotting Values to Collection Variables"). The EXTEND approach cannot be used to introduce an invalid assortment.

Where is it Defined?

A neighbourhood type is an assortment type defined in a PL/SQL block. If the square is in an independent or bundle subprogram, it is only available in the square and is stored in the data set. (See "Settled, Package, and Standalone Subprograms" for more information on independent and bundle subprograms.)

A public thing is an assortment type defined in a bundle detail. You can use the bundle name (package name.type name) to refer to it from outside the bundle. It is stored in the information base until the bundle is dropped.

An assortment type characterized at diagram level is an independent sort. You make it with the "Make TYPE Statement". It is put away in the information base until you drop it with the "DROP TYPE Statement".

4.5 INTRODUCTION TO VARYING ARRAYS

A varying array (variable-size exhibit) is a cluster whose number of components can range from zero (empty) to the maximum size declared.

Use the linguistic structure variable name to get to a component of a varying array variable (index). File has a bottom bound of 1 and an upper bound of the current number of components. As you add or remove components, the upper bound shifts, but it never exceeds the maximum size. When you save and retrieve a varying array from the data base, its records and component requests remain constant.

NOTES

Figure 4.12 depicts the Grades varying array variable, which has a maximum size of 10 and seven components. Grades(n) refers to the Grades nth component. Grades can't go higher than 7, and they can't go lower than 10.

Figure 4.12 Varying array of Maximum Size 10 with 7 Elements

Fig 4.12 *Varying array of Maximum Size 10 with 7 Elements*

A varying array variable is stored as a single article in the data collection. If a varying array variable is less than 4 KB in size, it stays inside the table of which it is a segment; otherwise, it lives outside the table but in the same tablespace.

An invalid assortment is a varying array variable that has not been initialised. It should be instated, either by leaving it empty or by assigning a non-NULL value to it. See "Assortment Constructors" and "Allocating Values to Collection Variables" for more details.

This model describes a neighbourhood VARRAY type, declares a variable of that type (with a function Object (), and describes a strategy for printing the varying array. The approach is invoked multiple times in the model: after establishing the variable, after changing the upsides of two components solely, and after using a function Object () to change the upsides, all things considered.

Suitable Uses for varying arrays

A varying array is suitable when:

- You know the greatest number of components.
- You normally access the components successively.

Since you should store or recover all components simultaneously, a varying array may be unreasonable for enormous quantities of components.

4.5.1 Creation of Varying Arrays

This model defines a neighborhood VARRAY type, announces a variable of that type (with a function Object (), and defines a printing strategy for the varying array. The approach is invoked three times throughout the model after introducing the variable, after modifying the upsides of two components independently, and finally, after using a function Object() to change the upsides.

```
DECLARE
 TYPE Foursome IS VARRAY(4) OF VARCHAR2(15); — VARRAY type
 — varying array variable initialized with constructor:
 team Foursome := Foursome('John', 'Mary', 'Alberto', 'Juanita');
 PROCEDURE print_team (heading VARCHAR2) IS
 BEGIN
```

NOTES

```
  DBMS_OUTPUT.PUT_LINE(heading);
   FOR i IN 1..4 LOOP
    DBMS_OUTPUT.PUT_LINE(i || '.' || team(i));
   END LOOP;
  DBMS_OUTPUT.PUT_LINE('—');
 END;
BEGIN
 print_team('2001 Team:');
 team(3) := 'Pierre'; — Change values of two elements
 team(4) := 'Yvonne';
 print_team('2005 Team:');
 — Invoke constructor to assign new values to varray variable:
 team := Foursome('Arun', 'Amitha', 'Allan', 'Mae');
 print_team('2009 Team:');
END;
/
Result:
2001 Team:
1.John
2.Mary
3.Alberto
4.Juanita
—
2005 Team:
1.John
2.Mary
3.Pierre
4.Yvonne
—
2009 Team:
1.Arun
2.Amitha
3.Allan
4.Mae
—
```

NOTES

4.5.2 Maintaining of Varying Arrays

VARRAY represents the variable-sized cluster.

A VARRAY is single-layered assortments of components with similar information type. Dissimilar to a cooperative cluster and settled table, a VARRAY consistently has a decent number of elements(bounded) and never has holes between the components (not meager).

Pronounce a VARRAY type

To pronounce a VARRAY type, you utilize this language structure:

TYPE type_name IS VARRAY(max_elements)

OF element_type [NOT NULL];

In this statement:

type_name is the sort of the VARRAY.

max_elements is the most extreme number of components permitted in the VARRAY.

NOT NULL determines that the component of the VARRAY of that sort can't have NULL components. Note that a VARRAY variable can be invalid, or uninitialized.

element_type is the sort of components of the VARRAY type's variable.

To make a VARRAY type which is open universally in the information base, not simply in your PL/SQL code, you utilize the accompanying sentence structure:

Make [OR REPLACE] TYPE type_name AS | IS

VARRAY(max_elements) OF element_type [NOT NULL];

In this announcement, the OR REPLACE adjusts existing sort while keeping all current awards of advantages.

Proclaim and instate VARRAY factors

When you made your own VARRAY type, you can proclaim a VARRAY case of that sort by referring to the VARRAY type. The essential language structure for VARRAY assertion is:

varray_name type_name [:= type_name(...)];

In this linguistic structure:

The varray_name is the name of the VARRAY.

The type_name is the VARRAY type.

The type_name(...) is the constructor of the VARRAY type, which acknowledges a comma-isolated rundown of components as contentions. It has a similar name as the VARRAY type.

Note that prior to utilizing a VARRAY variable, you should introduce it. Any other way, you will get the accompanying mistake:

To introduce a VARRAY variable to an unfilled assortment (zero components), you utilize the accompanying language structure:

varray_name type_name := type_name();

Code language: SQL (Structured Query Language) (sql)

To indicate components for the VARRAY variable while introducing it, you can utilize this grammar:

varray_name type_name := type_name(element1, element2, ...);

Code language: SQL (Structured Query Language) (sql)

Getting to exhibit components

To get to an exhibit component you utilize the accompanying grammar:

varray_name(n);

Code language: SQL (Structured Query Language) (sql)

n is the file of the component, which starts with 1 and closures with the max_elements the most extreme number of components characterized in the VARRAY type.

On the off chance that n isn't in the reach (1, max_elements), PL/SQL raises the SUBSCRIPT_BEYOND_COUNT mistake.

Basic PL/SQL VARRAY model

The accompanying square shows a straightforward instance of utilizing VARRAY factors:

```
Announce
  TYPE t_name_type IS VARRAY(2)
     OF VARCHAR2(20) NOT NULL;
  t_names t_name_type := t_name_type('John','Jane');
  t_enames t_name_type := t_name_type();
Start
  — introduce to a vacant cluster
   dbms_output.put_line("The number of components in t_enames " ||
t_enames.COUNT);
  — introduce to a variety of a components
   dbms_output.put_line("The number of components in t_names " ||
t_names.COUNT);
END;
/
```

Code language: SQL (Structured Query Language) (sql)

In this model:

In the first place, pronounce a VARRAY of VARCHAR(2) with two components:

```
TYPE t_name_type IS
  VARRAY(2) OF VARCHAR2(20) NOT NULL;
```

Code language: SQL (Structured Query Language) (sql)

NOTES

Then, proclaim a VARRAY variable and introduce it to a VARRAY of two components:

t_names t_name_type := t_name_type('John','Jane');

Code language: SQL (Structured Query Language) (sql)

Then, at that point, proclaim another VARRAY variable and instate it to a vacant cluster:

t_enames t_name_type := t_name_type();

Code language: SQL (Structured Query Language) (sql)

From that point forward, utilize the COUNT technique to get the quantity of components in the VARRAY t_enames and show it.

dbms_output.put_line("The number of components in t_enames " || t_enames.COUNT);

Code language: SQL (Structured Query Language) (sql)

At long last, utilize a similar COUNT technique to get the quantity of components in the VARRAY t_names and print it out.

dbms_output.put_line("The number of components in t_names " || t_names.COUNT);

Code language: SQL (Structured Query Language) (sql)

Note that you can relegate a VARRAY to another utilizing the accompanying punctuation:

varray_name := another_varray_name;

Code language: SQL (Structured Query Language) (sql)

For instance:

t_enames := t_names;

Code language: SQL (Structured Query Language) (sql)

PL/SQL coppies all individuals from t_names to t_enames.

2) PL/SQL VARRAY of records model

See the accompanying model:

Pronounce

```
  TYPE r_customer_type IS RECORD(
    customer_name customers.NAME%TYPE,
    credit_limit customers.credit_limit%TYPE
   );
  TYPE t_customer_type IS VARRAY(2)
     OF r_customer_type;
  t_customers t_customer_type := t_customer_type();
```

Start

```
  t_customers.EXTEND;
```

NOTES

```
    t_customers(t_customers.LAST).customer_name := 'ABC Corp';
    t_customers(t_customers.LAST).credit_limit := 10000;
    t_customers.EXTEND;
    t_customers(t_customers.LAST).customer_name := 'XYZ Inc';
    t_customers(t_customers.LAST).credit_limit := 20000;
        dbms_output.put_line('The number of clients is ' ||
t_customers.COUNT);
END;
/
```

Code language: SQL (Structured Query Language) (sql)

To begin with, characterize a record type that incorporates two fields client name and credit limit.

```
TYPE r_customer_type IS RECORD(
  customer_name customers.name%TYPE,
  credit_limit customers.credit_limit%TYPE
);
```

Code language: SQL (Structured Query Language) (sql)

Then, pronounce a VARRAY kind of the record r_customer_type with the size of two:

```
TYPE t_customer_type IS VARRAY(2)
  OF r_customer_type;
```

Code language: SQL (Structured Query Language) (sql)

Then, at that point, proclaim a VARRAY variable of the VARRAY type t_customer_type:

```
t_customers t_customer_type := t_customer_type();
```

Code language: SQL (Structured Query Language) (sql)

From that point onward, utilize the EXTEND technique to add an example to t_customers and the LAST strategy to add a component toward the finish of the VARRAY t_customers

```
t_customers.EXTEND;
t_customers(t_customers.LAST).customer_name := 'ABC Corp';
t_customers(t_customers.LAST).credit_limit := 10000;
t_customers.EXTEND;
t_customers(t_customers.LAST).customer_name := 'XYZ Inc';
t_customers(t_customers.LAST).credit_limit := 20000;
```

Code language: SQL (Structured Query Language) (sql)

At long last, utilize the COUNT strategy to get the quantity of components in the cluster:

NOTES

dbms_output.put_line('The number of clients is ' || t_customers.COUNT);

Code language: SQL (Structured Query Language) (sql)

Here is the result of the square:

The quantity of clients is 2

Code language: SQL (Structured Query Language) (sql)

3) Adding components to VARRAY from a cursor model

The accompanying model uses a cursor to recover five clients who have the most noteworthy credits from the clients table and add information to a VARRAY:

clients table

```
Announce
  TYPE r_customer_type IS RECORD(
    customer_name customers.name%TYPE,
    credit_limit customers.credit_limit%TYPE
  );
  TYPE t_customer_type IS VARRAY(5)
     OF r_customer_type;
  t_customers t_customer_type := t_customer_type();
  CURSOR c_customer IS
    SELECT NAME, credit_limit
     FROM clients
    Request BY credit_limit DESC
     Bring FIRST 5 ROWS ONLY;
Start
  — bring information from a cursor
  FOR r_customer IN c_customer LOOP
    t_customers.EXTEND;
   t_customers(t_customers.LAST).customer_name := r_customer.name;
                t_customers(t_customers.LAST).credit_limit :=
r_customer.credit_limit;
  END LOOP;

  — show all clients
  FOR l_index IN t_customers .FIRST..t_customers.LAST
  Circle
    dbms_output.put_line(
       'The client ' ||
      t_customers(l_index).customer_name ||
```

```
        ' has a credit of ' ||
      t_customers(l_index).credit_limit
     );
  END LOOP;
END;
/
```

Code language: SQL (Structured Query Language) (sql)

In this model:

In the first place, pronounce a record type, a VARRAY kind of the record with 5 components, and a VARRAY variable of that VARRAY type:

```
TYPE r_customer_type IS RECORD(
  customer_name customers.name%TYPE,
  credit_limit customers.credit_limit%TYPE
);
TYPE t_customer_type IS VARRAY(5)
  OF r_customer_type;
t_customers t_customer_type := t_customer_type();
```

Code language: SQL (Structured Query Language) (sql)

Second, proclaim a cursor that recovers 5 clients with the most noteworthy credits:

```
CURSOR c_customer IS
  SELECT name, credit_limit
  FROM clients
  Request BY credit_limit DESC
  Get FIRST 5 ROWS ONLY;
```

Code language: SQL (Structured Query Language) (sql)

Third, process the cursor and add every component to the VARRAY t_customers:

```
FOR r_customer IN c_customer LOOP
  t_customers.EXTEND;
  t_customers(t_customers.LAST).customer_name := r_customer.name;
  t_customers(t_customers.LAST).credit_limit := r_customer.credit_limit;
END LOOP;
```

Code language: SQL (Structured Query Language) (sql)

At last, emphasize over the components of the VARRAY t_customers and print out the client name and credit:

```
FOR l_index IN t_customers .FIRST..t_customers.LAST
```

```
Circle
  dbms_output.put_line(
     'The client ' ||
    t_customers(l_index).customer_name ||
     ' has a credit of ' ||
    t_customers(l_index).credit_limit
   );
  END LOOP;
```

Code language: SQL (Structured Query Language) (sql)

Here is the result:

```
The client General Mills has a credit of 179916.92
The client NextEra Energy has a credit of 141953.76
The client Southern has a credit of 127665.21
The client Jabil Circuit has a credit of 113340.75
The client Progressive has a credit of 94989.78
```

Code language: SQL (Structured Query Language) (sql)

Erase components

To erase all components of a VARRAY, you utilize the DELETE technique:

```
varray_name.DELETE;
```

Code language: SQL (Structured Query Language) (sql)

To eliminate one component from the finish of a VARRAY, you utilize the TRIM strategy:

```
varray_name.TRIM;
```

Code language: SQL (Structured Query Language) (sql)

To eliminate n components from the finish of a VARRAY, you utilize the TRIM(n) technique:

```
varray_name.TRIM(n)
```

4.5.3 Introduction to Nested Tables

Nested tables are unbounded, single-layered collections of homogeneous components.

To begin, a settled table is single-layered, suggesting that each line contains a single segment of data, similar to a one-aspect display.

A settled table, on the other hand, is limitless. It means that the number of components in a set table is predetermined.

Third, homogeneous components mean that the information types of all components of a resolved table are similar.

It's been observed that a settled table is first thick. It can, however, become depleted due to component expulsion.

NOTES

Pronouncing a table variable that has been settled

It takes two people to pronounce a settled table.

To begin with, pronounce the settled table sort utilizing this grammar:

```
TYPE nested_table_type
IS TABLE OF element_datatype [NOT NULL];
```

Code language: SQL (Structured Query Language) (sql)

Then, at that point, proclaim the settled table variable dependent on a settled table sort:

```
nested_table_variable nested_table_type;
```

Code language: SQL (Structured Query Language) (sql)

It is feasible to make a settled table sort situated in the data set:

```
Make [OR REPLACE] TYPE nested_table_type
  IS TABLE OF element_datatype [NOT NULL];
```

Code language: SQL (Structured Query Language) (sql)

To drop a sort, utilize the accompanying DROP TYPE articulation:

```
DROP TYPE type_name [FORCE];
```

Code language: SQL (Structured Query Language) (sql)

Instating a settled table

At the point when you proclaim a settled table variable, it is instated to NULL.

To introduce a settled table, you can utilize a constructor work. The constructor work has a similar name as the sort:

```
nested_table_variable := nested_table_type();
```

Code language: SQL (Structured Query Language) (sql)

You can likewise pronounce a settled table and introduce it in one stage utilizing the accompanying grammar:

```
nested_table_variable nested_table_type := nested_table_type();
```

Code language: SQL (Structured Query Language) (sql)

Add components to a settled table

To add a component to a settled table, you first utilize the EXTEND strategy:

```
nested_table_variable.EXTEND;
```

Code language: SQL (Structured Query Language) (sql)

Then, at that point, utilize the task administrator (:=) to add a component to the settled table:

```
nested_table_variable := component;
```

Code language: SQL (Structured Query Language) (sql)

To add numerous components, you utilize the EXTEND(n) strategy, where n is the quantity of components that you need to add:

NOTES

```
nested_table_variable.EXTEND(n);
nested_table_variable := element_1;
nested_table_variable := element_2;
..
nested_table_variable := element_n;
```

Code language: SQL (Structured Query Language) (sql)

Getting to components by their files

To get to a component at a predetermined record, you utilize the accompanying sentence structure:

```
nested_table_variable(index);
```

Code language: SQL (Structured Query Language) (sql)

Emphasize over the components of a settled table

Settled tables have the FIRST and LAST strategies that return the first and last files of components separately.

In this manner, you can utilize these techniques to repeat over the components of a settled table utilizing a FOR circle:

```
FOR l_index IN nested_table_variable.FIRST..nested_table_variable.LAST
Circle
  — access component
END LOOP;
```

Code language: SQL (Structured Query Language) (sql)

Assembling everything

We'll utilize the clients table from the example information base for the show:

clients table

The accompanying model shows how to utilize a cursor to get the initial 10 client names, add the client names to a settled table, and repeat over the components:

```
Announce
  — announce a cursor that return client name
  CURSOR c_customer IS
     SELECT name
     FROM clients
     Request BY name
     Bring FIRST 10 ROWS ONLY;
  — announce a settled table sort
  TYPE t_customer_name_type
    IS TABLE OF customers.name%TYPE;
  — announce and introduce a settled table variable
  t_customer_names t_customer_name_type := t_customer_name_type();
```

```
Start
  — populate client names from a cursor
  FOR r_customer IN c_customer
  Circle
    t_customer_names.EXTEND;
    t_customer_names(t_customer_names.LAST) := r_customer.name;
  END LOOP;
  — show client names
  FOR l_index IN t_customer_names.FIRST..t_customer_names.LAST
  Circle
    dbms_output.put_line(t_customer_names(l_index));
  END LOOP;
END;
```

Code language: SQL (Structured Query Language) (sql)

How about we analyze the model exhaustively.

To begin with, proclaim a cursor that profits the initial 10 in order arranged client names.

```
CURSOR c_customer IS
      SELECT name
      FROM clients
      Request BY name
       Get FIRST 10 ROWS ONLY;
```

Code language: SQL (Structured Query Language) (sql)

Then, pronounce a settled table sort:

```
TYPE t_customer_name_type
  IS TABLE OF customers.name%TYPE;
```

Code language: SQL (Structured Query Language) (sql)

Then, at that point, announce a settled table variable and introduce it utilizing the settled table constructor:

```
t_customer_names t_customer_name_type := t_customer_name_type();
```

Code language: SQL (Structured Query Language) (sql)

From that point forward, get client names from the cursor and add them to the settled table:

```
FOR r_customer IN c_customer LOOP
  t_customer_names.EXTEND;
  t_customer_names(t_customer_names.LAST) := r_customer.name;
END LOOP;
```

Code language: SQL (Structured Query Language) (sql)

At long last, repeat over the components of the settled table and show each:

NOTES

```
FOR l_index IN t_customer_names.FIRST..t_customer_names.LAST
Circle
  dbms_output.put_line(t_customer_names(l_index));
END LOOP;
```

Code language: SQL (Structured Query Language) (sql)

Here is the result:

```
3M
ADP
AECOM
AES
AIG
AT&T
AbbVie
Abbott Laboratories
Advance Auto Parts
Aetna
```

Check Your Progress

7. What is an ORD?
8. Write one feature of OOP.
9. What does data abstraction refer to?
10. What do you understand by message passing?
11. How is the concept of dynamic binding implemented?
12. Define the term nested table.

4.6 ANSWERS TO 'CHECK YOUR PROGRESS'

1. A database trigger helps in maintaining an organization's database in such a manner that updates and validates the data without executing the PL/SQL code explicitly.
2. The only difference between stored procedures and triggers is that triggers are run automatically by the database whenever the events such as insert, update and delete operations occur.
3. The different types of triggers are as follows:
   - `Row` triggers
   - `Statement` triggers
   - `Before` and `After` triggers
   - `Instead` of triggers

- Triggers on system events and user events

4. To compile a trigger at SQL prompt, the following statement is as follows:

```
SQL> @emp_history
```

5. The general form of IF statement in trigger are as follows:
   - `If Inserting Then`
   - `If Deleting Then`
   - `If Updating Then`
6. The two states in which a trigger exists are enabled state and disabled state.
7. A data set administration framework (DBMS) made up of both a social data set (RDBMS) and an item organised data set is known as an article social information base (ORD) (OODBMS).
8. The programs written in OOP are easy to maintain and extend as new objects can be easily added to the existing system whenever required without modifying the other objects.
9. The data abstraction refers to the act of representing the essential features without including the background details or explanations.
10. Message passing is a process of interaction between different objects in a program.
11. The concept of dynamic binding is implemented with the help of inheritance and runtime polymorphism (virtual functions).
12. Nested tables are unbounded, single-layered collections of homogeneous components.

4.7 SUMMARY

- A database trigger is a stored procedure that is fired when an insert, update or delete statement is issued against the associated table.
- Database trigger can be used to enforce integrity constraints (e.g. check the referenced data to maintain referential integrity) across the clients in a distributed database.
- A database trigger has three parts, namely trigger statement, trigger body and trigger restriction.
- Database triggers get executed or fired when a DML operation is performed on its associated table.
- A Form trigger execute or fires when the user navigates between fields on the screen or presses a key at run time.
- Database triggers can manipulate data stored in Oracle tables via SQL commands.
- Triggers can be enabled, disabled or dropped.
- A trigger does not apply on stored data; it works during the transaction on its associated table.

- A data set administration framework (DBMS) made up of both a social data set (RDBMS) and an item organised data set is known as an article social information base (ORD) (OODBMS).
- A programming paradigm describes the structure of a program. In other words, it determines how the instructions are placed in the program. Each programming language follows one or the other programming paradigm.
- In an unstructured programming paradigm, all the instructions of a program were written one after another in a single function and hence were suitable for writing only small and simple programs.
- In a structured programming, programs are divided into different procedures (also known as functions, routines or subroutines) and each procedure contains a set of instructions that performs a specific task. It follows the top-down approach.
- The structured programming paradigm had certain limitations that led to the development of object oriented paradigm.
- In an object oriented programming paradigm, programmers define not only the data but also the operations (functions) that can be performed on it, together under a single unit. It follows the bottom-up approach.
- OOP is based on certain important concepts that include objects, classes, abstraction, encapsulation, inheritance and polymorphism.
- An object is a unit of structural and behavioural modularity that contains a set of properties (or data) as well as the associated functions.
- A class is defined as a user defined data type that contains the entire set of similar data and the functions that objects possess. The process of creating objects from a class is known as instantiation.
- Static approaches are invoked based on the article type rather than the occasion. For tasks that are global to the type and do not require referencing the information of a single item example, you use a static technique.
- Nested tables are unbounded, single-layered collections of homogeneous components.

4.8 KEY TERMS

- **Trigger:** It is a PL/SQL code block that automatically triggers (runs) an event.
- **Create trigger:** It is a statement that is used to create a trigger in a disabled state.
- **Programming paradigm:** It is a programming methodology, which describes the structure of a program.
- **Unstructured programming paradigm:** It refers to the Instructions of a program written one after another in a single function and are suitable for writing only small and simple programs.

- **Structured programming:** It is a powerful programming tool, also known as procedural programming, provides an easy approach of writing complex programs.
- **Objects:** It refers to small, self-contained and modular units with a well-defined boundary.
- **State:** It is one of the possible conditions that an object can exist in and is represented by its characteristics or attributes or data.
- **Class:** It refers to user defined data type that contains the entire set of similar data and the functions that the objects possess.
- **Instantiation:** It is the process of creating objects from a class.
- **Abstraction:** It is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things.
- **Encapsulation:** It is the technique of binding or keeping data and functions that operate on them together in a single unit.
- **Inheritance:** It is the process whereby an object of a class acquires characteristics from the object of another class.
- **Abstract class:** It is a class that provides only the interface of one or more functions and not their implementations.
- **Nested tables:** It is unbounded, single-layered collections of homogeneous components.

4.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you understand by trigger in PL/SQL? How is it useful?
2. How will you differentiate database triggers from integrity constraints?
3. What types of trigger can be declared in PL/SQL?
4. Differentiate between `BEFORE` and `AFTER` triggers.
5. Is it possible to grant a trigger?
6. How can a trigger be deleted? Give an example for the same.
7. How are the new OOP features helpful?
8. What are data members?
9. Why is OOP paradigm developed?
10. What is operator overloading?
11. What is function overloading?
12. What do you mean by the object view?
13. What are the nested tables?

Long-Answer Questions

1. Discuss the various events on which you could fire a trigger.
2. Does the Statement Triggers work in the same manner as Row Trigger? Justify your answer.
3. What are the advantages of trigger over stored procedures and functions? Explain.
4. What are the stages of trigger compilation? Discuss.
5. What are the general forms of `IF` statements you could use in trigger?
6. Define the various trigger states; also explain the impact of those states on trigger execution.
6. Discuss the concept of object oriented programming paradigm with the help of examples.
7. Describe the approach of OOP principles with the help of illustrations and examples.
8. Explain objects, classes, abstraction, encapsulation, inheritance and polymorphism with the help of illustrations.
9. What is message passing? Explain.
10. Explain the benefits of OOP with the help of illustrations and examples.
11. Discuss about the manipulation data through object view with the help of diagram.
12. Describe the creating and maintaining varying array by giving appropriate examples.

4.10 FURTHER READING

Snowdon. 1998. *Oracle Programming With Visual Basic.* India: John Wiley & Sons.

Ying Bai. 2021. *Oracle Database Programming with Visual Basic.*NET. India: Wiley-IEEE Press. First Edition.

Byrla. 2017. *Oracle Database 12C*. India: McGraw Hill Education. First Edition.

P.S Deshpande. 2011. *SQL & PL/SQL for Oracle 11g.* India: Dreamtech Press.

UNIT 5 INTRODUCTION TO WEB ENABLED DATABASE AND DATABASE ADMINISTRATION

NOTES

Structure

5.0 INTRODUCTION

Oracle C++ Call Interface (OCCI) includes classes and methods for performing operations on large objects (LOBs). LOBs are either internal or external depending on their location with respect to the database. Internal LOBs are stored inside database tablespaces in a way that optimizes space and enables efficient access. Internal LOBs use copy semantics and participate in the transactional model of the server. You can recover internal LOBs in the event of transaction or media failure, and any changes to an internal LOB value can be committed or rolled back. In other words, all the ACID properties that pertain to using database objects also pertain to using internal LOBs. External LOBs (BFILES) are large binary data objects stored in operating system files outside database tablespaces. These files use reference semantics. Apart from conventional secondary storage devices, such as hard disks, BFILEs may also be located on tertiary block storage devices, such as CD-ROMs, PhotoCDs and DVDs.

Database administrators (DBAs) use specialized software to store and organize data. The role may include capacity planning, installation, configuration, database design, migration, performance monitoring, security, troubleshooting, as well as backup and data recovery. Database administration is the function of managing and maintaining database management systems (DBMS) software. Mainstream DBMS software such as Oracle, IBM DB2 and Microsoft SQL Server need ongoing management. As such, corporations that use DBMS software often hire specialized information technology personnel called database administrators or DBAs.

NOTES

In this unit you will study about the available data types, specifying storage for LOB data, introduction to web enabled database, role of SQL, understand the role of Java and WebDB, introduction to web architecture and database administration, creating a database, creating and managing rollback segment, backup and recovery.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the available data types
- Describe the specifying storage for LOB data
- Discuss the Concept of web enabled database
- Explain the role of SQL
- Describe the role of Java and WebDB
- Describe web architecture and database administration
- Discuss the creation a database
- Explain creating and managing rollback segment
- Explain the meaning of backup and recovery

5.2 USING LARGE OBJECTS

Enormous Objects (LOBs) are a bunch of datatypes that are intended to hold a lot of information. A LOB can hold up to a most extreme size going from 8 terabytes to 128 terabytes relying upon how your information base is arranged. Putting away information in LOBs empowers you to get to and control the information productively in your application.

Use Large Objects

This segment presents various sorts of information that you experience when creating applications and talks about which sorts of information are appropriate for enormous articles.

On the planet today, applications should manage the accompanying sorts of information:

Straightforward organized information.

This information can be coordinated into straightforward tables that are organized dependent on business rules.

Complex organized information

Such an information is complicated in nature and is appropriate for the item social highlights of the Oracle data set like assortments, references, and client characterized types.

Semi-organized information

Such an information has a sensible construction that isn't regularly deciphered by the data set. For instance, a XML report that is handled by your application or an outer help, can be considered as semi-organized information. The information base gives advances like Oracle XML DB, Advanced Queueing, and Messages to help your application work with semi-organized information.

Unstructured information

Such an information isn't separated into more modest intelligent constructions and isn't ordinarily deciphered by the data set or your application. A visual picture put away as a parallel record is an illustration of unstructured information.

Huge articles are reasonable for these last two sorts of information: semi-organized information and unstructured information. Enormous articles highlights permit you to store these sorts of information in the data set just as in working framework records that are gotten to from the data set.

Can store unstructured and semi-organized information in a productive way.

Is enhanced for a lot of information.

Involving LOBs for Semi-organized Data

Semi-organized data can be found in archival records, such as XML reports or word processor documents. When saved in a data collection, these records include information in a logical structure that can be processed or decrypted by an application, and they aren't broken down into smaller, more coherent chunks. In applications containing semi-organized data, a lot of character information is typically used. For storing and controlling this type of data, the datatypes Character Large Object (CLOB) and National Character Large Object (NCLOB) are perfect. Twofold File objects can also be used to store character data (BFILE datatypes). You can utilise BFILEs to stack read-only data from working framework documents into CLOB or NCLOB cases, which you can subsequently alter in your app.

Involving LOBs for Unstructured Data

Unstructured data is impossible to break down into discrete components. Information on a representative, for example, can be organised into a name, which is recorded as a string; an identification, such as an ID number, compensation, or something similar. In contrast, a snapshot is made up of a continuous stream of 1s and 0s. These components are used to switch pixels on and off so that the image on a display may be seen, but they are not split into a more suitable structure for data set storage.

Text, realistic graphics, still video cuts, full motion video, and sound waveforms are all examples of unstructured data that will be big in size. A representative record of two or three hundred bytes can represent even little amounts of mixed media information.

The SQL datatypes BLOB (Binary Massive Object) and BFILE (Binary File) are appropriate for large volumes of unstructured double data (Binary File object).

Why Not Use LONGs?

LONG and LOB datatypes are both backed by the same data set. Because of the added benefits that LOBs give, you should utilise them instead of LONGs in your

NOTES

current applications wherever practical. You may quickly convert your existing apps that use LONG sections to use LOB segments with LONG-to-LOB migration.

5.3 AVAILABLE DATATYPES

A datatype is assigned to each value limited by Oracle Database. The datatype of a value enables an appropriate sequence of characteristics to be applied to the item. Because of these characteristics, Oracle treats potential gains of one datatype differently than possible gains of another. Potential benefits of the NUMBER datatype, for example, can be added, but not those of the RAW datatype.

When creating a table or pack, you should choose a datatype for each of its sections. You should choose a datatype for all of your conflicts when creating a technique or setting aside a limit. These datatypes outline the range of attributes that each segment or disagreement may possess. DATE pieces, for example, are unable to detect the value February 29 (unless it is a leap year) or the attributes 2 or 'SHOE.' Every value that must be placed in a segment expects the fragment's datatype. If you insert '01-JAN-98' into a DATE section, for example, Oracle interprets the '01-JAN-98' character string as a DATE regard after confirming that it denotes a meaningful date.

Consume Datatype

A fixed-length character string is represented by the CHAR datatype. Prophet guarantees that all CHAR section attributes have the length indicated by size. Oracle clear pads the value to part length if you install a value that is more restrictive than the segment length. If you try to insert a value that is excessively long for the segment, Oracle will produce an error.

A CHAR fragment's default length is 1 byte, with a maximum length of 2000 bytes. A 1-byte string can be inserted into a CHAR (10) segment, but the string must first be clear padded to 10 bytes.

When creating a table with a CHAR fragment, you must specify the section length in bytes. The BYTE qualifier is the same as the default qualifier. If you use the CHAR qualifier, such as CHAR(10 CHAR), you must specify the section length in characters. A person is, in fact, a code point in the data base's individual set. Depending on the informational index individual set, its size might range from 1 to 4 bytes. The semantics governed by the NLS LENGTH SEMANTICS limit, which has a default of byte semantics, are overridden by the BYTE and CHAR qualifiers. Oracle suggests setting length semantics with the NLS LENGTH SEMANTICS limit for performance reasons, and using the BYTE and CHAR qualifiers only when absolutely necessary.

NCHAR Datatype

The NCHAR datatype is a datatype that only accepts Unicode characters. You represent the segment length in characters when you create a table with an NCHAR part. When you create your informational collection, you describe the public individual set. The public individual set definition determines the best length of a portion. The character datatype NCHAR's width conclusions indicate the number

of characters. The maximum permitted section size is 2000 bytes. Oracle clear pads the value to section length if you introduce a value that is more restricted than the part length. A CHAR regard cannot be implanted into an NCHAR segment, nor can an NCHAR regard be implanted into a CHAR section.

VARCHAR2 Datatype

A variable-length character string is represented by the VARCHAR2 datatype. When you create a VARCHAR2 part, you give it the most ridiculous number of bytes or characters it can hold. If the advantage doesn't outperform the section's most outrageous length of the portion, Prophet stores every value in the segment definitively as you reveal it. If you try to install a value that outperforms the predetermined length, Oracle will return a failure.

For a VARCHAR2 area, you should show an extremely long length. This upper limit should be around 1 byte, however the genuine string set aside is allowed to be 0 in length. To offer the most absurd length in characters rather than bytes, use the CHAR qualifier, for example VARCHAR2 (10 CHAR). A person is, in fact, a code point in the data base's individual set. The Burn and BYTE qualifiers override the NLS LENGTH SEMANTICS limit, which is set to bytes by default. Oracle advises that you utilise the NLS LENGTH SEMANTICS limit to set length semantics, and that you only use the BYTE and CHAR qualifiers when the limit is not sufficient. The optimal VARCHAR2 data length is 4000 bytes. Prophet mulls about VARCHAR2's nonpadded assessment semantics.

Floating Point Numbers

Floating point numbers can have a decimal point anywhere between the first and last digits, or they can have no decimal point at all. On the other side, a sort might be used to construct the span later. Because the number of digits that can appear after the decimal point is not limited, a scale view is not appropriate for floating point numbers.

In the way the attributes are handled inside Oracle Database, matched floating point numbers shift from NUMBER. The values are taken care of, including NUMBER's decimal exactness. All literals that fall inside the range and precision of NUMBER are treated conclusively as NUMBER. Due to the fact that literals are sent with decimal precision, literals are handled unequivocally (the digits 0 through 9). Double accuracy is used to store paired drifting point numbers (the digits 0 and 1). A capacity plot of this size can't handle all attributes with decimal accuracy. When a value is changed from decimal to paired accuracy, the mistake that occurs is usually spread when the value is restored back to decimal accuracy. A model like this is the precise 0.1.

5.3.1 Specifying Storage for LOB Data

Creating LOB-Containing Tables

Use the criteria shown in the following regions when causing tables that contain LOBs:

Persistent LOBs being set to NULL or Empty

You can make a consistent LOB, such as a LOB fragment in a database or

NOTES

a LOB property in a represented article type, NULL or void:

Changing the value of a Persistent LOB to NULL: There is no locater for a LOB that is set to NULL. The table line, not the locater, takes care of a NULL value. In the case of any residual data types, this is a similar cycle.

Setting an Empty Persistent LOB: An unfilled LOB set aside in a table, on the other hand, is a zero-length LOB with a locater. As a result, if you SELECT from an empty LOB section or attribute, you'll get a locater, which you may use to populate the LOB with data using OCI or PL/SQL (DBMS LOB) maintained specified conditions.

Changing the value of a Persistent LOB to NULL

In cases when you don't have the LOB data at the time of the INSERT, you may need to set a constant LOB value to NULL after installing the line, or use a SELECT declaration, similar to the going with, to determine if the LOB carries a NULL value:

SELECT COUNT (*) FROM print_media WHERE ad_graphic IS NOT NULL;

SELECT COUNT (*) FROM print_media WHERE ad_graphic IS NULL;

The truth is that on a NULL LOB, you can't choose a limit from the retained programmed conditions. These restrictions merely function with a locater, and if the LOB segment is NULL, there is no locater in the segment.

Setting an Empty Persistent LOB

A consistent LOB can be associated with EMPTY rather than NULL. Doing so allows you to obtain a locater for the LOB model without having to fill it with data. Use the SQL work EMPTY BLOB() or EMPTY CLOB() in the INSERT clarification to make a driving forward LOB EMPTY:

INSTALL VALUES (EMPTY BLOB()) INTO a table;

Instead of calling a subsequent SELECT attestation, you may include the RETURNING condition to acquire the LOB locater in one movement:

Start Install INTO a table; Report Lob loc BLOB RETURNING blob col INTO Lob loc; / VALUES (EMPTY BLOB()) * Use the locater Lob loc to fill the BLOB with data for now */ END;

Introducing LOBs

You can use the going with INSERT explanation to provide the LOBs in print media:

Insert INTO print_media VALUES (1001, EMPTY_CLOB(), EMPTY_CLOB(), NULL,

EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL);

This makes ad sourcetext, ad fltextn, ad composite, and ad photo have an empty value, and ad graphic is NULL.

Adding Attributes and Persistent LOB Columns to a Value

The LOB section or LOB attributes can be applied to a value that contains more than 4G bytes of data, which was the previous cutoff before release 10.2.

Changing the value of BFILEs to NULL or a File Name

The value of a BFILE can be NULL or a filename. You can use the BFILENAME() function to accomplish this.

"Initialization and BFILENAME".

Limit on First Extent of a LOB Segment

Any segment's essential level requires something like two squares (if FREELIST GROUPS was 0). That is, the piece's fundamental degree size should be around two squares. Throws segments are recognisable due to the fact that they require at least three squares to begin with. If you try to build a LOB segment in a very strong word reference regulated tablespace with starting = 2 squares, it succeeds because areas in long-term word reference controlled tablespaces can override the tablespaces' default accumulating setting.

However, accepting that uniform secretly administered tablespaces or word reference regulated tablespaces of the fleeting kind, or secretly directed momentary tablespaces have a degree size of 2 squares, then, LOB parts can't be made in these tablespaces. This is in light of the fact that in these tablespace types, degree sizes are fixed and the default storing setting of the tablespaces isn't ignored.

Picking a LOB Column Data Type

While picking a data type, contemplate the going with three subjects:

Hurls Compared to LONG and LONG RAW Types records the likenesses and differentiations between LOBs, LONGs, and LONG RAW sorts.

Table 5.1 *LOBs Vs. LONG*

LOB Data Types	Long Data Types
You can store various LOBs in a single line.	You can store simply a solitary LONG or LONG RAW in every section.
It represents the basic data types.	It consumes more memory in Long Data Type.
A LOB can be up to 128 terabytes or more in size dependent upon your square size.	A LONG or LONG RAW model is limited to 2 gigabytes in size
For inline LOBs, the data base stores LOBs that are not by and large around 4000 bytes of data in the table area.	By virtue of a LONG or LONG RAW the entire worth is taken care of in the table area.
Exactly when you access a LOB fragment, you can choose to bring the locater or the data.	Right when you access a LONG or LONG RAW, the entire worth is returned.
There is more conspicuous flexibility in controlling data in a subjective, piece-wise way with LOBs. Heaves can be gotten to aimlessly offsets.	Less flexibility in controlling data in a discretionary, piece-wise way with LONG or LONG RAW data.LONGs ought to be gotten to from the beginning to the best region.
Taking care of Varying-Width Character Data in LOBs.	Replication in both area and scattered conditions is inconceivable with a LONG or LONG RAW.

NOTES

Changing width character data in CLOB and NCLOB data types is taken care of in an inside setup that is practical with UCS2 Unicode character set association. This ensures that there is no limit loss of character data in a changing width plan. Similarly the going with accepting you are using LOBs to store varying width character data:

You can make tables containing CLOB and NCLOB fragments whether or not you use a changing width CHAR or NCHAR data base individual set.

You can make a table containing a data type that has a CLOB quality whether or not you use a contrasting width CHAR informational index individual set.

Certain Character Set Conversions with LOBs

Character set modifications are undoubtedly conducted while deciphering commencing with one specific set then onto the next for CLOB and NCLOB circumstances employed in OCI (Oracle Call Interface) or any of the programmed conditions that access OCI convenience.

When stacking to a CLOB or NCLOB, the DBMS LOB.LOADCLOBFROMFILE API performs a precise transformation from combined data to character data. LOB APIs do not play out particular changes from combined data to character data, with the exception of DBMS LOB.LOADCLOBFROMFILE.

When you use the DBMS LOB.LOADFROMFILE API to populate a CLOB or NCLOB, for example, you are actually loading the LOB with binary data from a BFILE. For the time being, you should modify the character set of the BFILE data before calling DBMS LOB.LOADFROMFILE.

Prophet Database Globalization Support Guide for more detail on character set changes.

The informational collection individual set can't be changed from a single byte to a multibyte character set expecting there are populated customer portrayed CLOB sections in the data base tables. The public individual set can't be changed some place in the scope of AL16UTF16 and UTF8 expecting there are populated customer described NCLOB areas in the data base tables.

Toss Storage Parameters

This portion summarizes LOB accumulating characteristics to contemplate when arranging tables with LOB storing. For a discussion of SECUREFILE limits:

"Using CREATE TABLE with SecureFiles LOBs"

"Using ALTER TABLE with SecureFiles LOBs"

Inline and Out-of-Line LOB Storage

Fling portions store locaters that reference the space of the certified LOB regard. Dependent upon the segment properties you decide when you make the table, and depending the size of the LOB, genuine LOB regards are taken care of either in the table line (inline) or outside of the table line (misguided).

Discard regards are put misguided when any of the going with conditions apply:

If you explicitly demonstrate DISABLE STORAGE IN ROW for the LOB amassing stipulation when you make the table.

Expecting the size of the LOB is more critical than around 4000 bytes (4000 short structure control information), paying little regard to the LOB accumulating properties for the section.

Accepting you update a LOB that is taken care of misguided and the ensuing LOB is not actually about 4000 bytes, it is at this point set aside misguided.

Discard regards are put inline when any of the going with conditions apply:

Right when the size of the LOB set aside in the given line is nearly nothing, around 4000 bytes or less, and you either unequivocally decide ENABLE STORAGE IN ROW or the LOB accumulating condition when you make the table, or when you don't show this limit (which is the default).

Exactly when the LOB regard is NULL (paying little psyche to the LOB storing properties for the segment).

Using the default LOB accumulating properties (inline limit) can consider better informational index execution; it avoids the overhead of making and managing misguided amassing for more unassuming LOB regards. Expecting LOB regards set aside in your data base are occasionally minimal in size, then, using inline storing is recommended.

A LOB locater reliably exists for any LOB model paying little notice to the LOB storing properties or LOB regard - NULL, empty, or regardless.

Expecting that the LOB is made with DISABLE STORAGE IN ROW properties and the BasicFiles LOB holds any data, then, somewhere around one CHUNK of misguided additional room is used; regardless, when the size of the LOB isn't actually the CHUNK size.

Accepting that a LOB section is instated with EMPTY_CLOB() or EMPTY_BLOB(), then, no LOB regard exists, not even NULL. The line holds a LOB locater figuratively speaking. No additional LOB accumulating is used.

Toss accumulating properties don't impact BFILE portions. BFILE data is continually taken care of in working system archives outside the informational index.

Portraying Tablespace and Storage Characteristics for Persistent LOBs

When describing LOBs in a table, you can unequivocally show the tablespace and limit credits for every consistent LOB portion.

To make a BasicFiles LOB, the BASICFILE expression is optional anyway is proposed for clearness, as shown in the going with model:

```
Make TABLE ContainsLOB_tab (n NUMBER, c CLOB)
    throw (c) STORE AS BASICFILE segname (TABLESPACE lobtbs1
CHUNK 4096
                PCTVERSION 5
                NOCACHE LOGGING
              Limit (MAXEXTENTS 5)
               );
```

NOTES

For SecureFiles, the SECUREFILE expression is significant, as shown in the going with model (tolerating TABLESPACE lobtbs1 is ASSM):

Make TABLE ContainsLOB_tab1 (n NUMBER, c CLOB)

throw (c) STORE AS SECUREFILE sfsegname (TABLESPACE lobtbs1

Support AUTO

Store LOGGING

Limit (MAXEXTENTS 5)

);

:

There are no tablespace or limit characteristics that you can show for external LOBs (BFILEs) as they are not taken care of in the informational collection.

Accepting that you ought to change the LOB storing limits on a current LOB area, then, use the ALTER TABLE ... MOVE clarification. You can change the RETENTION, PCTVERSION, CACHE, NOCACHE LOGGING, NOLOGGING, or STORAGE settings. You can moreover change the TABLESPACE using the ALTER TABLE ... MOVE announcement.

Consigning a LOB Data Segment Name

As shown in the in the past model, demonstrating a name for the LOB data segment makes for an essentially more instinctual work area. While scrutinizing the LOB data word reference sees USER_LOBS, ALL_LOBS, DBA_LOBS (see Oracle Database Reference), you see the LOB data part that you picked rather than system created names.

Toss Storage Characteristics for LOB Column or Attribute

Toss amassing characteristics that can be shown for a LOB segment or a LOB trademark join the going with:

TABLESPACE

PCTVERSION or RETENTION

Save/NOCACHE/CACHE READS

LOGGING/NOLOGGING

Piece

Engage/DISABLE STORAGE IN ROW

Limit

For most customers, defaults for these limit credits are sufficient. To change LOB amassing, then, contemplate the going with rules.

"Limit stipulation" and "Upkeep limit" in Oracle Database SQL Language Reference

TABLESPACE and LOB Index

Best execution for LOBs can be cultivated by demonstrating amassing for LOBs in a tablespace not exactly equivalent to the one used for the table that contains the LOB. If a wide scope of LOBs are gotten to constantly, then, it may moreover be important to decide an alternate tablespace for each LOB fragment or trademark to reduce contraption struggle.

The LOB record is an inward development that is solidly associated with LOB amassing. This recommends that a customer may not drop the LOB record and change it.

The structure sorts out which tablespace to use for LOB data and LOB record dependent upon your detail in the LOB storing condition:

Expecting that you don't decide a tablespace for the LOB data, then, the tablespace of the table is used for the LOB data and rundown.

Expecting you show a tablespace for the LOB data, then, both the LOB data and document use the tablespace not set in stone.

Tablespace for LOB Index in Non-Partitioned Table

While making a table, accepting that you show a tablespace for the LOB document for a non-distributed, then, your detail of the tablespace is ignored and the LOB record is co-arranged with the LOB data. Allocated LOBs really do avoid the LOB record semantic construction.

Demonstrating an alternate tablespace for the LOB accumulating sections engages a decrease in debate on the tablespace of the table.

PCTVERSION

Right when a BasicFiles LOB is changed, one more type of the BasicFiles LOB page is made to assist consistent with perusing of prior types of the BasicFiles LOB regard.

PCTVERSION is the level of all used BasicFiles LOB data space that can be involved by old variations of BasicFiles LOB data pages. At the point when old types of BasicFiles LOB data pages start to have more than the PCTVERSION proportion of used BasicFiles LOB space, Oracle Database endeavors to recuperate the old structures and reuse them. Accordingly, PCTVERSION is the percent of used BasicFiles LOB data obstructs that is available for framing old BasicFiles LOB data.

PCTVERSION has a default of 10 (%), something like 0, and a constraint of 100.

To close what regard PCTVERSION should be set to, ponder the going with:

How habitually BasicFiles LOBs are invigorated?

How habitually the invigorated BasicFiles LOBs are scrutinized?

"Recommended PCTVERSION Settings" gives a couple of rules to concluding a sensible PCTVERSION regard given an update level of 'X'.

NOTES

Table 5.2: *Recommented PCTVERSION Settings*

BasicFiles LOB Update Pattern	BasicFiles LOB Read Pattern	PCTVERSION
Invigorates X% of LOB data	Scrutinizes invigorated LOBs	X%
Invigorates X% of LOB data	Scrutinizes LOBs anyway not the invigorated LOBs	0%
Invigorates X% of LOB data	Scrutinizes both invigorated and non-revived LOBs	2X%
Never invigorates LOB	Gets LOBs	0%

If your application requires a couple of BasicFiles LOB invigorates concurrent with significant examines of BasicFiles LOB fragments, then, ponder including a higher motivating force for PCTVERSION, for instance, 20%.

Setting PCTVERSION to twice the default regard allows every one of the more free pages to be used for old variations of data pages. Since gigantic inquiries may require consistent examines of BasicFiles LOB fragments, it very well may be important to hold old transformations of BasicFiles LOB pages. For the present circumstance, BasicFiles LOB accumulating may create considering the way that the data base doesn't reuse free pages strongly.

If steady BasicFiles LOB events in your application are made and made just a solitary time and are basically examined only a brief time frame later, then, revives are uncommon. For the present circumstance, consider including a lower an impetus for PCTVERSION, for instance, 5% or lower.

The more uncommon and more unobtrusive the BasicFiles LOB revives are, the less space ought to be put something aside for old copies of BasicFiles LOB data. If current BasicFiles LOBs are known to be examined nobody in any case, you could safely set PCTVERSION to 0% considering the way that there would never be any pages needed for old types of data.

Parameter for BasicFiles LOBs

As a choice as opposed to the PCTVERSION limit, you can decide the RETENTION limit in the LOB storing state of the CREATE TABLE or ALTER TABLE announcement. Doing in that capacity, plans the LOB section to store old types of LOB data for some time, rather than using a level of the table space. For example:

Make TABLE ContainsLOB_tab (n NUMBER, c CLOB)

toss (c) STORE AS BASICFILE segname (TABLESPACE lobtbs1 CHUNK 4096

Support

NOCACHE LOGGING

Limit (MAXEXTENTS 5)

);

The RETENTION limit is expected for use with UNDO features of the data base, similar to Flashback Versions Query. Exactly when a LOB segment has the RETENTION property set, old versions of the LOB data are held for how long controlled by the UNDO_RETENTION limit.

Fix SQL isn't enabled for LOB segments everything considered with various data types. You should set the RETENTION property on a LOB segment to use Undo SQL on LOB data.

You can't set the value of the RETENTION limit explicitly. How much an optimal chance for upkeep of LOB transformations in directed by the UNDO_RETENTION limit.

Utilization of the RETENTION limit is only maintained in Automatic Undo Management mode. You ought to organize your table for use with Automatic Undo Management before you can set RETENTION on a LOB area. ASSM is required for LOB RETENTION to be basically for BasicFiles LOBs. The RETENTION limit of the SQL (in the STORE AS stipulation) is unobtrusively dismissed expecting the BasicFiles LOB lives in a MSSM tablespace.

The LOB accumulating stipulation can demonstrate RETENTION or PCTVERSION, yet all the equivalent not both.

Prophet Database Advanced Application Developer's Guide for additional information on using flashback features of the informational index.

Prophet Database SQL Language Reference for nuances on LOB amassing condition sentence structure.

Upkeep Parameter for SecureFiles LOBs

Deciding the RETENTION limit for SecureFiles shows that the data base supervises solid read data for the SecureFiles storing capably, considering components like the UNDO strategy for the informational collection.

Show MAX if the informational index is in FLASHBACK mode to confine the size of the LOB UNDO upkeep in bytes. If you decide MAX, then, you ought to similarly demonstrate the MAXSIZE stipulation in the storage_clause.

Demonstrate AUTO to hold UNDO satisfactory for consistent read purposes in a manner of speaking. This is the default.

Demonstrate NONE expecting no UNDO is required for either unsurprising read or flashback purposes.

The default RETENTION for SecureFiles is AUTO.

Save/NOCACHE/CACHE READS

When causing tables that to contain LOBs, use the save decisions as shown by the standards in Table 5.3, "When to Use CACHE, NOCACHE, and CACHE READS":

Table 5.3 When to Use CACHE, NOCACHE, and CACHE READS

Store Mode	Read	Write
Store READS	Routinely	Once or every so often
Store	Consistently	Consistently
NOCACHE (default)	Once or inconsistently	Never

Hold/NOCACHE/CACHE READS: LOB Values and Buffer Cache

Hold: Oracle places LOB pages in the help store for speedier access.

NOCACHE: As a limit in the STORE AS explanation, NOCACHE confirms that LOB regards are not brought into the help save.

Store READS: LOB regards are brought into the pad hold simply during read and not during create exercises.

NOCACHE is the default for both SecureFiles and BasicFiles LOBs.

Using the CACHE decision results in additional created execution when scrutinizing and forming data from the LOB section. Regardless, it may conceivably age other non-LOB pages out of the pad hold imprudently.

LOGGING/NOLOGGING Parameter for BasicFiles LOBs

[NO]LOGGING has an equivalent application concerning including LOBs as it achieves for other table exercises. In the average case, accepting the [NO]LOGGING arrangement is blocked, then, this suggests that neither NOLOGGING nor not really set in stone and the logging quality of the table or table fragment defaults to the logging normal for the tablespace in which it stays.

For LOBs, there is a further choice depending upon how CACHE is indicated.

Store is shown and [NO]LOGGING proclamation is blocked. LOGGING is subsequently executed (in light of the fact that you can't have CACHE NOLOGGING).

Not really set in stone and [NO]LOGGING explanation is rejected. The cycle defaults in basically the same manner as it achieves for tables and separated tables. That is, the [NO]LOGGING regard is gotten from the tablespace in which the LOB segment stays.

The going with issues should similarly be recalled.

Throws Always Generate Undo for LOB Index Pages

Whether or not LOGGING or NOLOGGING is set, LOBs never make rollback information (fix) for LOB data pages since old LOB data is taken care of in structures. Rollback information that is made for LOBs will overall be little since it is only for the LOB record page changes.

When LOGGING is Set Oracle Generates Full Redo for LOB Data Pages

NOLOGGING is wanted to be used when a customer can't muster enough willpower to care with respect to media recovery. Likewise, expecting the circle/tape/accumulating media crashes and burns, you can't recover your movements from the sign in light of the fact that the movements were seldom logged.

NOLOGGING is Useful for Bulk Loads or Inserts.

For instance, while stacking data into the LOB, expecting you can't muster the energy to care about re-attempt and can essentially start the store indeed if it miss the mark, set the LOB data piece amassing characteristics to NOCACHE NOLOGGING. This gives extraordinary execution to the hidden stack of data.

At whatever point you have completed the process of stacking data, if fundamental, use ALTER TABLE to change the LOB accumulating characteristics for the LOB data piece for standard LOB undertakings, for example, to CACHE or NOCACHE LOGGING.

Hold proposes that you furthermore get LOGGING.

LOGGING/FILESYSTEM_LIKE_LOGGING for SecureFiles LOBs

NOLOGGING or LOGGING has a practically identical application concerning including SecureFiles as LOGGING/NOLOGGING achieves for other table exercises. In the standard case, if the logging_clause is avoided, the SecureFiles gains its logging quality from the tablespace in which it lives. For the present circumstance, expecting that NOLOGGING is the default regard, the SecureFiles defaults to FILESYSTEM_LIKE_LOGGING.

Using the CACHE decision results in additional created execution when examining and making data from the LOB area. Regardless, it may potentially age other non-LOB pages out of the support hold thoughtlessly.

Store Implies LOGGING

For SecureFiles, there is a further choice depending upon how CACHE is demonstrated:

Not really settled and the LOGGING explanation is ignored, then, LOGGING is used.

Not really settled and the logging_clause is disregarded. Then, the cycle defaults in much the same way as it achieves for tables and isolated tables. That is, the LOGGING regard is gotten from the tablespace in which the LOB regard abides. If the tablespace is NOLOGGING, the SecureFiles defaults to FILESYSTEM_LIKE_LOGGING.

The going with issues should in like manner be recollected.

SecureFiles and an Efficient Method of Generating REDO and UNDO

This infers that Oracle Database concludes whether it is more useful to make REDO and UNDO for the change to a square, similar to stack squares, on the other hand expecting it delivers a variation and full REDO of the new square like BasicFiles LOBs.

FILESYSTEM_LIKE_LOGGING is Useful for Bulk Loads or Inserts

For instance, while stacking data into the LOB, expecting you can't muster the energy to care with respect to REDO and can essentially start the pile by and by if it misfires, set the LOB data area storing characteristics to FILESYSTEM_LIKE_LOGGING. This gives incredible execution to the hidden pile of data.

At whatever point you have completed the process of stacking data, if fundamental, use ALTER TABLE to change the LOB accumulating characteristics for the LOB data area for normal LOB exercises. For example, to CACHE or NOCACHE LOGGING.

Protuberance

A protuberance is something like one Oracle blocks. You can decide the piece size for the BasicFiles LOB while making the table that contains the LOB. This looks at to the data size used by Oracle Database while getting to or changing the LOB regard. Some part of the piece is used to store structure related information and the rest stores the LOB regard. The API you are using has a limit that benefits how much space used in the LOB protuberance to store the LOB regard. In PL/SQL use DBMS_LOB.GETCHUNKSIZE. In OCI, use OCILobGetChunkSize().

Picking the Value of CHUNK

At the point when the value of CHUNK is picked (when the LOB section is made), it can't be changed. Thusly, you should pick a value which further develops your ability and execution necessities. For SecureFiles, CHUNK is an admonition size and is obliged backward closeness purposes.

Space Considerations

The value of CHUNK does not significantly impact LOBs that are taken care of inline. This happens when ENABLE STORAGE IN ROW is set, and the size of the LOB locater and the LOB data isn't actually around 4000 bytes. Regardless, when the LOB data is taken care of misguided, it for the most part consumes room in results of the CHUNK limit.

5.3.2 Controlling and Selecting LOB Values

You can distribute that contain LOB segments. Appropriately, LOBs can take advantage of every one of the upsides of partitioning including the going with:

Throw parts can be spread between a couple tablespaces to change I/O load and to make support and recovery more reasonable.

Hurls in a distributed become less complex to stay aware of.

Tosses can be distributed keen social occasions to speed up methodology on LOBs that are gotten to altogether.

This section portrays a part of the habits wherein you can handle LOBs in allocated tables.

Partitioning a Table Containing LOB Columns

Throws are maintained in RANGE isolated, LIST divided, HASH distributed tables. Composite stack facilitated tables can in like manner have LOBs.

You can allocate table containing LOB areas using the going with procedures:

Right when the table is made using the PARTITION BY ... state of the CREATE TABLE decree.

Adding a bundle to a current table using the ALTER TABLE ... ADD PARTITION proclamation.

Exchanging allocations with a table that has isolated LOB segments using

the ALTER TABLE ... Exchange PARTITION articulation. that EXCHANGE PARTITION should be used when the two tables have a comparable accumulating credits, for example, the two tables store LOBs misguided.

Making LOB parts at the same time you make the table (in the CREATE TABLE verbalization) is proposed. Accepting you make fragments on a LOB area when the table is made, then, the section can hold LOBs set aside either inline or misguided LOBs.

Making an Index on a Table Containing Partitioned LOB Columns

To deal with the display of requests, you can make records on separated LOB portions. For example:

ON table_name (LOB_column_1, LOB_column_2, ...) LOCAL; that primary space and limit set up documents are maintained as for LOB fragments. Various types of records, for instance, unique records are not maintained with LOBs.

Moving Partitions Containing LOBs

You can move a LOB portion into a substitute tablespace. This is useful if the tablespace is at this point not gigantic enough to hold the fragment. To do accordingly, use the ALTER TABLE ... MOVE PARTITION stipulation. For example:

```
Change TABLE current_table MOVE PARTITION partition_name
  TABLESPACE destination_table_space
  Fling (column_name) STORE AS (TABLESPACE current_tablespace);
```

Isolating Partitions Containing LOBs

You can section a fragment containing LOBs into two comparably estimated portions using the ALTER TABLE ... SPLIT PARTITION condition. Doing as such permits you to place one or both new bundles in a new tablespace. For example:

```
Change TABLE table_name SPLIT PARTITION partition_name
  AT (partition_range_upper_bound)
  INTO (PARTITION partition_name,
    Bundle new_partition_name TABLESPACE new_tablespace_name
      Fling (column_name) STORE AS (TABLESPACE tablespace_name)
        ... ;
```

Mixing Partitions Containing LOBs

You can mix divides contain LOB portions using the ALTER TABLE ... Mix PARTITIONS stipulation. This technique is important for recuperating unused portion space. For example:

```
Adjust TABLE table_name
  Join PARTITIONS partition_1, partition_2
        INTO PARTITION new_partition TABLESPACE
new_tablespace_name
```

Hurl (column_name) store as (TABLESPACE tablespace_name)

... ;

Hurls in Index Organized Tables

NOTES

List Organized Tables (IOTs) support internal and outside LOB fragments. For the most part, SQL DDL, DML, and piece smart methodology on LOBs in IOTs produce comparable results as those for common tables. The really extraordinary case is the default semantics of LOBs during creation. The essential differentiations are:

Tablespace Mapping: By default, or with the exception of whenever showed regardless, the LOB data and rundown segments are made in the tablespace in which the fundamental key record pieces of the document composed table are made.

Inline as Compared to Out-of-Line Storage: By default, all LOBs in a rundown composed table made without a flood piece are taken care of misguided. Accordingly, if a document composed table is made without a flood segment, then, the LOBs in this table have their default amassing credits as DISABLE STORAGE IN ROW. Accepting you influentially endeavor to demonstrate an ENABLE STORAGE IN ROW explanation for such LOBs, then, SQL raises an error.

Of course, if a flood section has been demonstrated, then, LOBs in record facilitated tables exactly copy their semantics in ordinary tables (see "Describing Tablespace and Storage Characteristics for Persistent LOBs").

Representation of Index Organized Table (IOT) with LOB Columns

Contemplate the going with model:

Make TABLE iotlob_tab (c1 INTEGER PRIMARY KEY, c2 BLOB, c3 CLOB, c4

VARCHAR2(20))

Affiliation INDEX

TABLESPACE iot_ts

PCTFREE 10 PCTUSED 10 INITRANS 1 MAXTRANS 1 STORAGE (INITIAL 4K)

PCTTHRESHOLD 50 INCLUDING c2

Flood

TABLESPACE ioto_ts

PCTFREE 10 PCTUSED 10 INITRANS 1 MAXTRANS 1 STORAGE (INITIAL 8K) LOB (c2)

STORE AS lobseg (TABLESPACE lob_ts DISABLE STORAGE IN ROW

Bump 16384 PCTVERSION 10 CACHE STORAGE (INITIAL 2M)

Document lobidx_c1 (TABLESPACE lobidx_ts STORAGE (INITIAL 4K)));

NOTES

Executing these declarations achieves the creation of a record facilitated table iotlob_tab with the going with parts:

A fundamental key record segment in the tablespace iot_ts,

A flood data piece in tablespace ioto_ts

Fragments starting from section C3 being unequivocally taken care of in the flood data segment

Mass (section C2) data segments in the tablespace lob_ts

Mass (portion C2) record areas in the tablespace lobidx_ts

CLOB (area C3) data sections in the tablespace iot_ts

CLOB (area C3) document pieces in the tablespace iot_ts

CLOB (area C3) set aside in line by greatness of the IOT having a flood piece

Mass (area C2) unequivocally constrained to be taken care of misguided

:

If no flood had been shown, then, both C2 and C3 would have been taken care of misguided normally.

Other LOB features, for instance, BFILEs and contrasting person width LOBs, are moreover maintained in document facilitated tables, and their utilization is identical to for standard tables.

Impediments for LOBs in Partitioned Index-Organized Tables

Hurl fragments are maintained in range-, list-, and hash-distributed record composed tables with the going with impediments:

Composite distributed facilitated tables are not maintained.

Social and article separated record facilitated tables (allocated by reach, hash, or once-over) can hold LOBs set aside as follows; in any case, portion upkeep exercises, similar to MOVE, SPLIT, and MERGE are not maintained with:

VARRAY data types set aside as LOB data types

Hypothetical data types with LOB attributes

Settled tables with LOB types

To invigorate LOBs in a settled table, you should lock the section containing the LOB explicitly. To do accordingly, you ought to decide the FOR UPDATE arrangement in the subquery prior to invigorating the LOB regard.

5.4 INTRODUCTION TO WEB ENABLED DATABASE

Web administrations enable application-to-application communication over the Internet, regardless of platform, language, or data design. Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and

Integration (UDDI) are among the important fixes that have been adopted by the whole programming industry. The majority of web administrations allude to administrations that are executed and delivered on application servers at the center level. However, in diverse and dispersed environments, there is a growing requirement to access stored systems, such as information and metadata, via Web management interfaces.

Database Web administrations is a data-driven approach to dealing with Web administrations. It functions in the following two ways:

- Getting to data set assets as a Web administration
- Burning-through outside Web administrations from the data set

Prophet Database can get to Web administrations through PL/SQL bundles and Java classes conveyed inside the data set. Transforming Oracle Database into a Web specialist organization use interest in Java put away systems, PL/SQL bundles, predefined SQL inquiries, and information control language (DML). Then again, devouring outside Web administrations from the data set, along with coordination with the SQL motor, empowers Enterprise Information Integration.

Involving Oracle Database as Web Services Provider

Web Services make use of industry-standard tools to provide simple access to remote content and applications, regardless of the supplier's stage or location, as well as the execution and information architecture. Using conventional Web administration protocols, customer applications can query and retrieve information from Oracle Database and access stored methods. Oracle-specific information base network conventions are not used. In heterogeneous, diffused, and separated settings, this methodology is particularly helpful.

You can use a Web administration to access the data collection and use it as a specialist organisation. This allows you to leverage Oracle Database with existing or new SQL, PL/SQL, Java put away methodology, or Java classes.

5.4.1 Role of SQL

At the point when the SQL standard approval mode is empowered, object proprietors can utilize the SQL jobs office to direct honors.

SQL jobs are helpful for regulating honors when an information base has numerous clients. Jobs give an all the more impressive method for conceding honors to clients' meetings than to allow honors to every client of the information base, which effectively becomes dreary and blunder inclined when numerous clients are involved. Jobs don't all by themselves give better information base security, yet utilized accurately, they work with better security. Just the information base proprietor can make, award, renounce, and drop jobs. Notwithstanding, object proprietors can concede and disavow honors for those items to and from jobs, just as to and from individual clients and PUBLIC (all clients).

Making and allowing jobs

Only when SQL approval mode is enabled are jobs accessible (that is, the point at which the property derby.database.sqlAuthorization is set to TRUE).

Before jobs can be used, old information bases should be (hard) upgraded to Release 10.5 at the very least.

If SQL approval mode is enabled, the database owner can create jobs using the CREATE ROLE explanation. The owner of the information base could then use the GRANT explanation to assign a task to at least one client, PUBLIC, or another job.

If job B is conceded to work A, or is contained in a job C authorised to job A, then job A contains one more job B. A contained job's honours are gained by the containing jobs. As a result, the arrangement of awards acknowledged by job An is the sum of the honours granted to job An and the honours granted to any of job A's contained jobs.

For instance, assume the data set proprietor gave the accompanying assertions:

```
make job peruser;
make job updater;
make job taskLeaderA;
make job taskLeaderB;
make job projectLeader;
award peruser to updater;
award updater to taskLeaderA;
award updater to taskLeaderB;
award taskLeaderA to projectLeader;
award taskLeaderB to projectLeader;
```

The jobs would then have the accompanying regulation connections:

```
          peruser
            |
            v
          updater
         /      \
taskLeaderA taskLeaderB
         \      /
       projectLeader
```

For this situation, the projectLeader job contains the wide range of various jobs and has every one of their honors. Assuming the data set proprietor, denies updater from taskLeaderA, projectLeader actually contains that job through taskLeaderB.

The SYSCS_DIAG.CONTAINED_ROLES symptomatic table capacity can be utilized to decide the arrangement of contained jobs for a job.

NOTES

Cycles are not allowed in job awards. That is, assuming a job contains another job, you can't allow the compartment job to the contained job. For instance, the accompanying assertion would not be allowed:

award projectLeader to updater;

Setting jobs

At the point when a client initially associates with Derby, no job is set, and the CURRENT_ROLE work brings invalid back. During a meeting, the client can call the SET ROLE articulation to set the current job for that meeting. The job can be any job that has been conceded to the meeting's present client or to PUBLIC. To unset the current job, call SET ROLE with a contention of NONE. Whenever during a meeting, there is dependably a current client, however there is a current job provided that SET ROLE has been called with a contention other than NONE. Assuming that a current job isn't set, the meeting has just the honors conceded to the client straightforwardly or to PUBLIC.

For instance, on the off chance that the information base proprietor made and allowed the jobs displayed in the past meeting, a client would need to give a SET ROLE proclamation to have them produce results. Assume a client gave the accompanying assertion:

SET ROLE taskLeaderA;

Expecting that the data set proprietor had conceded the taskLeaderA job to the client, the client would be permitted to set the job as displayed and would have every one of the honors allowed to the taskLeaderA, updater, and peruser jobs.

To recover the current job identifier in SQL, call the CURRENT_ROLE work.

Inside put away strategies and capacities that contain SQL, the current job is on the approval stack. At first, inside a settled association, the current job is set to that of the calling setting. Upon get back from the put away methodology or capacity, the approval stack is popped, so the current job of the calling setting isn't impacted by any setting of the job inside the called system or capacity. Assuming the put away system opens more than one settled association, these all offer something similar (stacked) current job state. Any unique outcome set dropped of a put away technique sees the current job of the settled setting.

Giving honors to jobs

When a job has been made, both the data set proprietor and the article proprietor can give honors on tables and schedules to that job. You can allow the very honors to jobs that you can concede to clients. Allowing an honor to a job verifiably gives honors to all jobs that contain that job. For instance, assuming you award erase honors on a table to updater, each client in the updater, taskLeaderA, taskLeaderB, and projectLeader job will likewise have erase honors on that table, yet clients in the peruser job will not.

Disavowing honors from a job

Either the data set proprietor or the article proprietor can disavow honors from a job.

At the point when an honor is renounced from a job A, that honor is not generally held by job A, except if an in any case acquires that honor from a contained job.

Assuming an honor to an item is disavowed from job A, a meeting will lose that honor assuming it plays a current part set to An or a job that contains A, except if at least one of coming up next is valid:

The honor is conceded straightforwardly to the current client

The honor is conceded to PUBLIC

The honor is likewise conceded to one more job B in the current job's arrangement of contained jobs

The meeting's present client is the information base proprietor or the article proprietor

Renouncing jobs

The information base proprietor can utilize the REVOKE proclamation to deny a job from a client, from PUBLIC, or from another job.

At the point when a job is disavowed from a client, that meeting can don't really keep that job, nor would it be able to take on that job in a SET ROLE proclamation, except if the job is likewise conceded to PUBLIC. Assuming that job is the current job of a current meeting, the current honors of the meeting lose any additional honors got through setting that job.

The default drop conduct is CASCADE. In this way, all determined articles (requirements, perspectives and triggers) that depend on that job are dropped. Despite the fact that there might be alternate methods of satisfying that honor at the hour of the deny, any reliant articles are as yet dropped. This is an execution impediment. Any pre-arranged explanation that is possibly impacted will be checked again on the following execute. An outcome set that relies upon a job will stay open regardless of whether that job is disavowed from a client.

At the point when a job is disavowed from a job, the default drop conduct is additionally CASCADE. Assume you renounce job A from job B. Renouncing the job will repudiate all extra material honors acquired through A from B. Jobs that contain B will likewise lose those honors, except if An is as yet contained in some other job C conceded to B, or the honors get through another job. See Creating and conceding jobs for a model.

Dropping jobs

Just the information base proprietor can drop a job. To drop a job, utilize the DROP ROLE explanation.

Dropping a job successfully renounces all awards of this job to clients and different jobs.

5.4.2 Role of Java and WebDB

JDeveloper 3.0's Business Components for Java and WebDB are two different technologies. WebDB generates server-side PL/SQL packages that dynamically render HTML web pages based on database tables, stored procedures, etc.

NOTES

JDeveloper 3.0's Business Components for Java framework is a pure Java way to write middle-tier application logic for Java Applications or applets, or for JSPs or servlets. You can deploy the Business Components for Java to Oracle8i and an EJB or CORBA object, locally to the file system, or to Visibroker. You could certainly have the two technologies existing in the same database and have BC4J-based applications available from a website developed with WebDB as just straight links, but we do not support building WebDB components against the Business Components for Java framework. Building a JSP Web Application using the JDeveloper 3.0 wizards is the closest thing in terms of having an HTML front-end.

Web Application

Web apps are naturally disseminated applications, which means they are programmes that have a rapid surge in demand for multiple PCs and are distributed through an organisation or server. Web apps, in particular, are accessed using a web browser and are well-known for the ease with which the programming can be used as a client customer. For the project, the ability to update and maintain online apps without distributing and installing code on a potentially large number of consumer PCs is a critical reason for their popularity. Web apps are used for a variety of things, including web mail, online retail sales, chat sheets, blogs, and internet banking. A single web application can be accessed and used by a huge number of people. Web apps, like work area applications, are made up of numerous pieces and commonly contain smaller-than-expected projects, some of which have UIs and some of which don't require a graphical user interface (GUI) at all. Furthermore, online applications frequently necessitate the use of an additional markup or pre-arranging language, such as HTML, CSS, or JavaScript. Similarly, many applications only use the Java programming language, which is perfect because of its adaptability. A web application might be as simple as a page that displays the current date and time or as complex as a series of pages that allow you to search for the best airfare, accommodation, and car rental deals.

Web Applications with Java

Because there are too many Java innovations to discuss in one article, this one will focus on the most commonly used ones. The sheer number of innovations documented here can be overwhelming. In most cases, a web application is made up of only one page, which is created using the JavaServer Pages (JSP) technology. At times, you will be a part of at least three of these advancements. Regardless of how many you use, it is important to understand what is available and how you may include everyone in a web application.

Java Servlet API

You can use the Java Servlet API to describe HTTP-explicit classes. A servlet class extends the capabilities of servers with applications that use the solicitation reaction programming model to access them. Servlets, despite their ability to respond to requests, are most commonly used to extend the applications made possible by web servers. For instance, you could use a servlet to collect text input from a web-based structure and print it back to the screen in an HTML page and arrangement, or you could use a different servlet to compose the data to a record

or data set, all other things being equal. A servlet is a server-side application that doesn't have its own GUI or HTML (UI). Many people benefit from Java Servlet enhancements.

Customers might go in intricacy from straightforward HTML structures to refined Java innovation-based applets. The javax.servlet and javax.servlet.http bundles provide servlet-specific classes and connection points. The javax.servlet.http.HttpServlet conceptual class, which provides a structure for dealing with HTTP conventions, is expanded by HTML servlet classes.

JavaServer Pages Technology

The JavaServer Pages (JSP) breakthrough offers a new, faster way to create dynamic online content. JSP technology enables rapid development of server- and platform-agnostic electronic applications. The JSP innovation allows you to easily add snippets of servlet code to a text-based archive. A JSP page is a text-based archive that typically comprises two types of text:

Static data that can be shared in any text-based format, such as HTML, Wireless Markup Language (WML), or XML.

JSP innovation components control how the page generates dynamic content. The bundles javax.el, javax.servlet.jsp, javax.servlet.jsp.el, and javax.servlet.jsp.tagext are used to create JSP pages, but you won't need to import them directly. A JSP page can be as simple as a touch of HTML with one piece of JSP code and the page name's.jsp expansion.

You can, for example, create a site with JSP innovative pages that use a single line of code to include the header.html file, which includes the site route. In this way, if you modify a connection to a button in the route, you just have to change one document, and that document will be loaded into all of the pages on the site that have this code bit:

<%@ incorporate file="head.html" %>

If you're familiar with server-side incorporates, that line of code works similarly. Because this site page is already a JSP page, you might continue to add additional Java innovative code to create dynamic online content, such as surveys, structures, and methods for entering or retrieving data from a data set, and so on.

The JavaServer Pages Standard Tag Library (JSTL) is a collection of tags that are common to many JSP-based applications. Instead of combining labels from many vendors in your applications, you use a single standard set of labels. This standardization enables you to run your applications on any JSP holder that supports JSTL, increasing the likelihood of better label execution.

For dealing with stream control, JSTL offers iterator and restriction labels, labels for controlling XML reports, internationalization labels, labels for getting to data sets using SQL, and labels for commonly used capacities.

Javax.servlet.jsp.jstl.core, javax.servlet.jsp.jstl.fmt, javax.servlet.jsp.jstl.sql, and javax.servlet.jsp.jstl.tlv are the JSTL bundles you can use.

Java Message Service API

Informing is a method of establishing communication between programming

NOTES

components or applications. A dispersed office is an informative framework. As a result, a client who is informing can send and receive messages from other customers. Each consumer interacts with an information specialist who provides offices for creating, sending, receiving, and comprehending messages. The Java Message Service (JMS) API is a valuable asset for dealing with large company figuring challenges because it combines Java innovation with big business informing.

Throughout the course of a project, undertaking informing provides a solid, adjustable service for the exchange of business information. The JMS API adds a standard API and supplier structure to this, allowing Java programmers to create a wide range of message-based applications. An application that checks stock for an auto company is an example of how JMS could be used. When the stock level for an item falls below a certain level, the stock part can make an imprint on the processing plant component, allowing the production line to create more automobiles. The manufacturing plant portion can imprint on the parts in order for the industrial facility to collect the parts it requires. As a result, the components can send signals to their own stock, requesting that they update their inventories and arrange for new parts from providers, among other things.

The JMS API improves developer efficiency by defining a standard set of informing concepts and programming techniques that will be followed by all JMS innovation consistent informing frameworks.

JDBC API

The JDBC API allows you to call up data sets and SQL orders using Java programming techniques. When you need to get to the database, you can use the JDBC API in a servlet, a JSP innovation page, or a venture bean.

The JDBC API is divided into two sections: an application-level point of engagement for connecting to a database, and a specialist co-op point of interaction for adding a JDBC driver to the Java EE stage.

Java Persistence API

The Java Persistence API is a diligence solution based on Java innovation concepts. To deal with any barrier between an articles prepared model and a social data set, Diligence employs an item social planning method. There are three areas that make up Java's innovation prowess:

The Java Persistence API is a set of classes that allow you to store data

Object-social planning metadata is the question language.

Java Naming and Directory Interface

The Java Name and Directory Interface (JNDI) provides naming and indexing functionality, allowing applications to use a variety of naming and cataloguing services. It provides strategies for executing standard catalogue activities, such as assigning partner credits to objects and searching for objects based on their properties, to applications. A web application can use JNDI to save and retrieve any named Java innovation object, allowing apps to interface with a variety of inherited applications and frameworks.

Application clients, endeavor beans, and web parts have access to a JNDI naming environment through naming administrations. A naming climate allows an engineer to rework a portion without having to access or modify the source code. The present situation of the part is executed by a compartment and passed to the part as a JNDI naming setting.

Check Your Progress

1. What is Complex organized information?
2. Define the term available data types.
3. State about the space considerations.
4. What is Web administrations?

5.5 A BRIEF INTRODUCTION ABOUT DATABASE ADMINISTRATION

Prophet is a social data set. In a social data set, all information is put away in two-layered tables that are made out of lines and segments. The Oracle Database empowers you to store information, update it, and effectively recover it.

Prophet gives programming to make and deal with the Oracle information base. The data set comprises of physical and sensible designs in which framework, client, and control data is put away. The product that deals with the data set is known as the Oracle data set server. On the whole, the product that runs prophet and the actual data set are known as the Oracle information base framework.

You will get more familiar with the activity of the data set server and the design of the Oracle information base where they are pertinent to the exhibition of explicit data set administration errands.

Normal Oracle DBA Tasks

As an Oracle DBA, you can hope to be engaged with the accompanying errands:

Introducing Oracle programming

Making Oracle data sets

Performing redesigns of the information base and programming to new delivery levels

- Firing up and closing down the data set
- Dealing with the data set's stockpiling structures
- Overseeing clients and security
- Overseeing blueprint objects, like tables, files, and perspectives
- Making information base reinforcements and performing recuperation when fundamental

Proactively checking the information base's wellbeing and making a preventive or remedial move as required

NOTES

Checking and tuning execution

In a little to fair size data set climate, you may be the sole individual playing out these undertakings. In huge, endeavor conditions, the occupation is frequently split between a few DBAs, each with their own forte, for example, data set security or information base tuning.

Devices for Administering the Database

The goal of this book is to permit you to rapidly and effectively make an Oracle data set, and to give direction in essential information base organization.

Coming up next are a portion of the items, devices, and utilities you can use in accomplishing your objectives as a data set manager:

Prophet Universal Installer (OUI)

The Oracle Universal Installer introduces your Oracle programming and choices. It can naturally send off the Database Configuration Assistant to introduce an information base.

Information base Configuration Assistant (DBCA)

The Database Configuration Assistant makes an information base from layouts that are provided by Oracle, or you can make your own. It empowers you to duplicate a preconfigured seed data set, consequently saving the time and exertion of creating and altering an information base without any preparation.

Data set Upgrade Assistant

This Database Upgrade Assistant aides you through the redesign of your current data set to another Oracle discharge.

Prophet Net Manager

Net Manager guides you through your Oracle Net organization setup.

5.6 CREATING A DATABASE

Most of the database systems help you to create database objects using a WYSIWYG interface. Microsoft's SQL Server is supported by Enterprise Manager which provides a specific type of graphical representation of the database system that helps in browsing the databases for viewing the tables and their contents, altering the tables, updating the tables, etc. The `CREATE` commands in SQL* permit you to create database objects programmatically including the database and its tables. The following are the `CREATE` commands supported by SQL Server:

- `CREATE ACTION`
- `CREATE CACHE`
- `CREATE CELL CALCULATION`
- `CREATE CUBE`
- `CREATE DATABASE`
- `CREATE DEFAULT`
- `CREATE FUNCTION`

- CREATE INDEX
- CREATE MEMBER
- CREATE MINING MODEL
- CREATE PROCEDURE
- CREATE RULE
- CREATE SCHEMA
- CREATE SET
- CREATE STATISTICS
- CREATE TABLE
- CREATE TRIGGER
- CREATE UNIQUE CLUSTERED INDEX
- CREATE VIEW

The following are some of the most common CREATE commands syntax.

Creating a Database: The given command creates a specified database.

```
CREATE DATABASE database_name
```

Adding Arguments: There are various optional arguments that can be used with the CREATE DATABASE command. The support to specific arguments and their usage are documented in the database system.

Example: This following statement creates a database named as 'Payroll'. As no arguments are specified the database data files and transaction logs will be automatically created in the default location.

```
CREATE DATABASE Payroll
```

Now specify the name and location of the data file and transaction log of database by specifying the initial size of these files (with the SIZE argument), the maximum size it can grow to (with the MAXSIZE argument) and the growth increment of each file (using the FILEGROWTH) argument.

```
USE master
GO
CREATE DATABASE Payroll
ON
( NAME = Payroll_dat,
     FILENAME = 'C:\program files\microsoft sql
server\mssql\data\payrolldat.mdf',
   SIZE = 20MB,
   MAXSIZE = 70MB,
   FILEGROWTH = 5MB )
LOG ON
( NAME = 'Payroll_log',
     FILENAME = 'C:\program files\microsoft sql
server\mssql\data\payroll.ldf',
   SIZE = 10MB,
   MAXSIZE = 40MB,
   FILEGROWTH = 5MB )
```

NOTES

```
GO
```

Next step is the `CREATE TABLE`. Create a table using the following `CREATE TABLE` command.

```
CREATE TABLE table_name
(column_name_1 datatype,
column_name_2 datatype,
...
)
```

When you specify the correct data type with values it results in an empty table. To add data to the created table use an `INSERT` statement.

Modifying/Altering a Database

The `ALTER TABLE` statement is used to add, delete or modify records, rows and columns in an existing database table.

The following syntax is used to add a column in a table:

```
ALTER TABLE table_name
ADD column_name datatype
```

The following syntax is used to delete a column in a table:

```
ALTER TABLE table_name
DROP COLUMN column_name
```

The following syntax is used to change the data type of a column in a table:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

Searching a Database

After browsing the SQL Server you can search all the columns, rows or records of all the tables in a given database or any specific table for a specific keyword. A stored procedure 'SearchAllTables' can be used to search all tables in a specified database. It accepts a search string as input parameter for searching all char, varchar, nchar, nvarchar columns of all user created tables. The output of this stored procedure contains following two columns:

- The table name and column name in which the search string was initiated
- The actual content/value of the column

SQL Server also provides the functionality designed for applications and users for issuing full-text queries against character-based data in SQL Server tables. Before full-text queries run on a specified table, the database administrator creates a full-text index on the table which includes one or more character-based columns in the table. These columns can have any of the data types namely `char`, `varchar`, `nchar`, `nvarchar`, `text`, `ntext`, `image`, `xml`, `varbinary`, or `varbinary(max)`. To write full-text queries, SQL Server provides a set of full-text predicates (`CONTAINS` and `FREETEXT`) and rowset-valued functions (`CONTAINSTABLE` and `FREETEXTTABLE`). The users can perform a various types of full-text searches, such as searching on a single word or phrase, searching on a word or phrase close to another word or phrase,

or searching on synonymous forms of a specific word.

Matching

A query can be created in SQL Server to perform pattern matching using the wildcard characters. The usage of wildcards permits you to find data that matches a certain pattern before specifying it accurately. For example, if the wildcard '`C%`' is used then it matches any string starting with a capital C. To use a wildcard expression in a SQL query, use the WHERE and LIKE clause to specify it. The following is the example that shows the use of syntax:

```
SELECT *
FROM employees
WHERE last_name LIKE 'C%'
```

The following are the various wildcard expressions supported by SQL:

- The '`%`' wildcard matches zero or more characters of any type.
- The '`_`' wildcard matches exactly one character of any type.
- You can also specify a list of characters by enclosing them in square brackets. For example, the wildcard [aeiou] is used to match any vowel.
- You can also specify a range of characters by enclosing the specific range in square brackets. For example, the wildcard `[a-p]` will match any letter that is in between a to p.
- You can undo a range of specific characters by using the carat `(^)` character inside the square bracket as `[^aeiou]`. This will match any non-vowel character and not the vowels.

To perform advanced queries for matching you can combine these wildcards in complex patterns.

5.7 CREATING AND MANAGING ROLLBACK SEGMENTS

You must have the CREATE ROLLBACK SEGMENT framework honour to create rollback sections. The CREATE ROLLBACK SEGMENT articulation is used. The new rollback parts should be stored in a web-based tablespace. Rollback parts are often created as part of the data set generation content or cycle, but you may decide to add more in the future.

The accompanying subjects connecting with making rollback fragments are remembered for this part:

The CREATE ROLLBACK SEGMENT Statement

Here, we will discuss the following points:

- Bringing New Rollback Segments Online
- Setting Storage Parameters When Creating a Rollback Segment

The CREATE ROLLBACK SEGMENT Statement

The going with statement makes a rollback segment named RBS_02 in the

NOTES

RBSSPACE tablespace, using the default amassing limits of that tablespace. Since this is authentically not an equivalent server environment, it isn't critical to show PRIVATE or PUBLIC. The default is PRIVATE.

Make ROLLBACK SEGMENT rbs_02 TABLESPACE rbsspace;

For exact semantic construction, impediments, and endorsement necessities for the SQL clarifications used in administering rollback segments, see Oracle8i SQL Reference.

Bringing New Rollback Segments Online

New rollback segments are at first separated. You should give an ALTER ROLLBACK SEGMENT to welcome them on the web and make it available for use by trades of an event. See "Changing the ONLINE/OFFLINE Status of Rollback Segments" for additional information.

Accepting you make a private rollback segment, you should add the name of this new rollback part to the ROLLBACK_SEGMENTS instatement limit in the presentation limit record for the informational collection. Doing as such engages the private rollback section to be gotten by the event at model fire up. For example, in case two new private rollback segments are made and named RBS_01 and RBS_02, the ROLLBACK_SEGMENTS limit of the limit record should resemble the going with:

ROLLBACK_SEGMENTS = (RBS_01, RBS_02)

For information about the ROLLBACK_SEGMENTS instatement limit, see the Oracle8i Reference.

Setting Storage Parameters When Creating a Rollback Segment

Expect you expected to make a rollback piece RBS_01 with limit limits and ideal size set as follows:

The rollback piece is allotted a hidden level of 100K.

The rollback piece is allotted the second level of 100K.

The ideal size of the rollback area is 4M.

The base number of degrees and the amount of degrees at first assigned when the segment is made is 20.

The most outrageous number of degrees that the rollback area can allot, including the fundamental degree, is 100.

The going with declaration makes a rollback area with these qualities:

```
Disclose ROLLBACK SEGMENT rbs_01
            TABLESPACE rbsspace
            Limit (
      Early on 100K
      NEXT 100K
      Optimal 4M
      MINEXTENTS 20
      MAXEXTENTS 100);
```

You can't set a motivation for the limit PCTINCREASE. It is reliably 0 for rollback segments. The OPTIMAL accumulating limit is remarkable to rollback areas. For a discussion of limit limits see "Setting Storage Parameters" and the Oracle8i SQL Reference.

Set INITIAL and NEXT to a comparative worth to ensure that all degrees are a comparative size.

Make a colossal number of early on degrees to restrict the shot at dynamic extension. MINEXTENTS = 20 is a fair worth.

Do whatever it takes not to set MAXEXTENTS = UNLIMITED as this could cause futile increase of a rollback piece and conceivably of data archives due to a programming botch. Expecting you truth be told do decide UNLIMITED, realize that degrees for that piece ought to have something like 4 data blocks. Similarly, to change over a rollback segment whose MAXEXTENTS are confined to UNLIMITED, that rollback area can't be changed over expecting it has under 4 data blocks in any degree. To change over from limited to UNLIMITED, and have under 4 data blocks in a degree, your primary choice is to drop and indeed make the rollback segment.

You can drop rollback areas when the levels of a part become too isolated on plate, or the section ought to be moved in a substitute tablespace.

Before dropping a rollback part, guarantee that status of the rollback segment is OFFLINE. To drop is another status, you can't drop it. In case the status is INVALID, the part has at this point been dropped.

To drop a rollback part, use the DROP ROLLBACK SEGMENT clarification. You ought to have the DROP ROLLBACK SEGMENT system honor. The going with declaration drops the RBS1 rollback area:

DROP ROLLBACK SEGMENT rbs1;

5.7.1 When Rollback Information is required

Utilize the ROLLBACK explanation to fix work done in the current exchange or to physically fix the work done by an in-question disseminated exchange.

To move back your present exchange, no honors are important.

To physically move back an in-question disseminated exchange that you initially dedicated, you should have the FORCE TRANSACTION framework honor. To physically move back an in-question dispersed exchange initially dedicated by another client, you should have the FORCE ANY TRANSACTION framework honor.

Rollback Segment States

Rollback segment is actually similar to some other table partitions and record pieces, which involve degrees, moreover demand space and they get made in a tablespace. To play out any DML action against a table which is in a non-structure tablespace ('emp' in 'customer' tablespace), prophet requires a rollback segment from a non-system tablespace.

Exactly when a trade is going on a segment which is in non-system tablespace, then, Oracle needs a rollback piece which is furthermore in non-structure tablespace. This is the clarification we make an alternate tablespace just for the rollback segment.

Why rollback segments?

Fix the movements when a trade is moved back.

Ensure read consistency (various trades don't see dubious changes made to the informational collection).

Recover the informational index to a consistent state in case of frustrations.

There are two kinds of rollback segments

a) Private rollback segments (for single case informational collection).

b) Public rollback segments (for RAC or Oracle Parallel Server).

At the hour of informational collection creation, prophet normally makes a rollback piece by name SYSTEM in structure tablespace and it's ONLINE. This rollback segment can't be brought OFFLINE since Oracle needs it as long as DB is going. This can't be dropped as well.

Nobody however DBA can make the rollback areas (SYS is the owner) and cannot be accessible to normal customers.

SQL> CREATE [PUBLIC] ROLLBACK SEGMENT rbs-name

[TABLESPACE tbs-name]

Limit (INITIAL 20K NEXT 40K MINEXTENTS 2 MAXEXTENTS 50);

A rollback segment similarly has its own accumulating limits, and the rules in making RBS are:

1. We can't portray PCTINCREASE for RBS (not really as 0).

2. We should have something like 2 as MINEXTENTS.

Beside conventional amassing limits, rollback segments can in like manner be described with OPTIMAL. We better make these rollback sections in an alternate tablespace where no tables or records exist. We should jump at the chance to make assorted rollback segments in different tablespaces.

Anyway we have made rollback areas, we really want to bring them ONLINE, either by using init.ora or by using a SQL announcement.

To utilize/engage rollback pieces by having a limit in init.ora, ROLLBACK_SEGMENTS = R1,R2,R3

There is another strategy for bringing any rollback piece ONLINE, by DBA in Oracle is:

SQL> ALTER ROLLBACK SEGMENT rbs-name ONLINE;

Basically, we can moreover make it disengaged.

SQL> ALTER ROLLBACK SEGMENT rbs-name OFFLINE;

The amount of rollback segments that are needed in the data base are picked by the concurrent DML activity customers (number of trades). Most noteworthy

number of rollback parts can be portrayed in init.ora by MAX_ROLLBACK_SEGMENTS limit (until 9i).

NOTES

To execute CREATE ROLLBACK SEGMENT and ALTER ROLLBACK SEGMENT orders, UNDO_MANAGEMENT ought not be set or set to MANUAL.

The errand of the rollback part to a trade will be done using load changing technique (with respect to the amount of trades yet not the size of trades). A customer can request prophet for a particular rollback area for his trade.

SQL> SET TRANSACTION USE ROLLBACK SEGMENT;

SQL> SET TRANSACTION USE ROLLBACK SEGMENT rbs-name;

To consign a rollback section at meeting level

SQL> ALTER SESSION USE ROLLBACK SEGMENT rbs-name;

In a creation data set climate, we need to plan various kinds of rollback sections to help various sorts of exchanges. Typically, in the day hours we have more modest exchanges (information passage tasks) before the end-clients, and at the night we perform handling (group occupations), model clear sql technique refreshing tables and submitting toward the finish of the exchange.

What is Backup and Recovery?

A duplicate of information is referred to as a reinforcement. Significant parts of the data set, such as the control record and datafiles, can be included in this duplication. A reinforcement is a safeguard against unforseen data loss and application errors. If the first information is lost, a backup can be used to recreate it.

Actual reinforcements and reasonable reinforcements are the two types of reinforcements. Actual reinforcements are duplicates of actual data set documents, and they are the most important concern in a reinforcement and recovery method. The Recovery Manager (RMAN) utility or working framework utilities can be used to do actual reinforcements. Legitimate reinforcements, on the other hand, comprise cohesive information (for example, tables and put away methods) segregated with an Oracle utility and stored in a database. You can utilize intelligent reinforcements to enhance actual reinforcements.

There are two methods for performing Oracle reinforcement and recuperation: Recovery Manager and client oversaw reinforcement and recuperation.

Recuperation Manager (RMAN) is an Oracle utility that can back up, reestablish, and recuperate information base documents. It is a component of the Oracle data set server and doesn't need separate establishment.

You can likewise utilize working framework orders for reinforcements and SQL*Plus for recuperation. This strategy, likewise called client oversaw reinforcement and recuperation, is completely upheld by Oracle, in spite of the fact that utilization of RMAN is strongly suggested on the grounds that it is more hearty and extraordinarily works on organization.

NOTES

Regardless of whether you use RMAN or client oversaw strategies, you can enhance your actual reinforcements with coherent reinforcements of blueprint objects made utilizing the Export utility. The utility composes information from an Oracle data set to twofold working framework documents. You can later utilize Import to reestablish this information into a data set.

Reliable and Inconsistent Backups

A reliable reinforcement is one in which the records being upheld contain all progressions up to a similar framework change number (SCN). This implies that the documents in the reinforcement contain every one of the information taken from an equivalent moment. Dissimilar to a conflicting reinforcement, a predictable entire information base reinforcement doesn't need recuperation later it is reestablished.

A conflicting reinforcement is a reinforcement of at least one information base documents that you make while the data set is open or later the data set has closed down strangely.

Outline of Consistent Backups

A predictable reinforcement of an information base or a piece of a data set is a reinforcement wherein all read/compose datafiles and control records are checkpointed with a similar SCN.

The best way to make a predictable entire data set reinforcement is to close down the information base with the NORMAL, IMMEDIATE, or TRANSACTIONAL choices and make the reinforcement while the data set is shut. On the off chance that a data set is not closed down neatly, for instance, an occasion fizzles or you issue a SHUTDOWN ABORT explanation, then, at that point, the information base's datafiles are dependably conflicting—except if the data set is a perused just data set.

Prophet makes the control records and datafiles steady to a similar SCN during a data set designated spot. The just tablespaces in a steady reinforcement that are permitted to have more established SCNs are perused just and disconnected typical tablespaces, which are as yet predictable with the other datafiles in the reinforcement in light of the fact that no progressions have been made to them.

The significant point is that you can open the data set subsequent to reestablishing a steady entire data set reinforcement without requiring recuperation on the grounds that the information is now reliable: no activity is needed to make the information in the reestablished datafiles right. Henceforth, you can reestablish a year-old reliable reinforcement of your information base without performing media recuperation and without Oracle performing occasion recuperation. Obviously, when you reestablish a reliable entire information base reinforcement without applying re-try, you lose all exchanges that were made since the reinforcement was taken.

A predictable entire data set reinforcement is the main substantial reinforcement choice for information bases working in NOARCHIVELOG mode, on the grounds that in any case recuperation is essential for consistency. In

NOARCHIVELOG mode, Oracle doesn't chronicle the re-try logs, thus the required re-try logs probably won't exist on circle. A predictable entire reinforcement is additionally a legitimate reinforcement choice for information bases working in ARCHIVELOG mode. At the point when this kind of reinforcement is reestablished and filed logs are accessible, you have the choice of either opening the data set quickly and losing exchanges that were made since the reinforcement was taken, or applying the chronicled logs to recuperate those exchanges.

Outline of Inconsistent Backups

A conflicting reinforcement is a reinforcement wherein the records being supported don't contain every one of the progressions made at all the SCNs. All in all, a few changes are absent. This implies that the records in the reinforcement contain information taken from various moments. This can happen on the grounds that the datafiles are being altered as reinforcements are being taken. Prophet recuperation makes conflicting reinforcements steady by perusing all filed and online re-try logs, beginning with the most punctual SCN in any of the datafile headers, and applying the progressions from the logs once again into the datafiles.

Assuming the information base should be going 24 hours every day, seven days per week, then, at that point, you must choose the option to perform conflicting reinforcements of the entire data set. A reinforcement of online datafiles is called a web-based reinforcement. This necessitates that you run your data set in ARCHIVELOG mode.

Assuming you run the information base in ARCHIVELOG mode, then, at that point, you don't need to back up the entire data set at one time. For instance, assuming that your data set contains seven tablespaces, and assuming that you back up the control record just as an alternate tablespace every evening, then, at that point, in seven days you will back up all tablespaces in the data set just as the control document. You can think about this amazed reinforcement all in all data set reinforcement. In any case, assuming such an amazed reinforcement should be reestablished, then, at that point, you really want to recuperate utilizing all filed re-try logs that were made since the soonest reinforcement was taken.

Backing Up the Archived Logs and the Control File

Later open or conflicting shut reinforcements, Oracle suggests backing up totally documented logs created during the reinforcement, and afterward backing up the control record later the reinforcement finishes. Assuming you don't have all filed re-try logs delivered during the reinforcement, then, at that point, you can't recuperate the reinforcement since you don't have all the re-try records important to make it predictable.

How does Recovery works?

To restore a genuine support of a datafile or control record is to reproduce it and make it available to the Oracle data base server. To recover a restored datafile is to invigorate it by applying chronicled re-attempt logs and online re-attempt logs, that is, records of changes made to the informational collection later the support was taken. Accepting you use RMAN, then, you can in like manner

NOTES

recover datafiles with consistent fortifications, which are fortifications of a datafile that contain simply squares that changed later a previous slow support.

Later the central records are restored, media recovery ought to be begun by the customer. Media recovery incorporates various assignments to restore, progress forward, and roll back a support of informational collection records.

Media recovery applies chronicled re-attempt logs and online re-attempt logs to recover the datafiles. Whenever a change is made to a datafile, the change is first recorded in the web-based re-attempt logs. Media recovery explicitly applies the movements recorded in the on the web and chronicled re-attempt logs to the restored datafile to push it ahead.

To resolve issues achieved by intelligent data corruptions or customer botches, you can use Oracle Flashback. Prophet Flashback Database and Oracle Flashback Table let you quickly recover to a past time.

Figure 5.1 shows the major rule of help up, restoring, and performing media recovery on a data base.

Unlike media recovery, Oracle performs crash recovery and case recovery thus later a model dissatisfaction. Crash and event recovery recover a data base to its trade unsurprising state not well before model dissatisfaction. By definition, crash recovery is the recovery of an informational index in a lone model arrangement or an Oracle Real Application Clusters plan in which all events have crashed. Curiously, case recovery is the recovery of one bombarded event by a live model in an Oracle Real Application Clusters course of action.

The sort of recovery that takes a support and applies re-attempt is called media recovery. Media recovery revives a support to either to the current or to a predefined prior time. Usually, the articulation "media recovery" insinuates recovery of datafiles. Block media recovery is a more specific movement that you use when a few squares in no less than one records have been corrupted. In any case, you for the most part use a restored support to play out the recovery.

Complete Recovery

Complete recovery incorporates using re-attempt data or consistent fortifications got together with a support of an informational collection, tablespace, or datafile to revive it to the most recent second. It is called completed because Oracle applies all of the re-attempt changes contained in the archived and online logs to the support. Regularly, you perform complete media recovery later a media dissatisfaction hurts datafiles or the control record.

You can perform absolute recovery on a data base, tablespace, or datafile. If you are performing completed recovery generally data base, then, you ought to:

Mount the informational collection

Ensure that all datafiles you want to recover are on the web

Restore a support of the whole data base or the records you really want to recover

NOTES

Apply on the web or recorded re-attempt logs, or a mix of the two

In case you are performing completed recovery on a tablespace or datafile, then, you ought to:

Take the tablespace or datafile to be recovered disengaged in case the informational collection is open

Restore a support of the datafiles you want to recover

Apply on the web or archived re-attempt logs, or a blend of the two

Insufficient Recovery

Insufficient recovery, or second recovery, uses a support to convey a noncurrent type of the informational collection. With everything taken into account, you don't make any difference all of the re-attempt records made later the most recent support. You generally speaking perform divided recovery of the whole data base in the going with conditions:

Media disillusionment devastates a couple or all of the web-based re-attempt logs.

A customer botch causes data disaster, for example, a customer unintentionally drops a table.

You can't perform absolute recovery considering the way that a documented re-attempt log is missing.

You lose your current control record and ought to use a support control report to open the data base.

To perform insufficient media recovery, you ought to restore all datafiles from fortifications made before the opportunity to which you really want to recover and thereafter open the informational index with the RESETLOGS decision when recovery wraps up. The RESETLOGS movement makes one more appearance of the informational collection—thusly, a data base with one more surge of log progression numbers starting with log plan 1.

Tablespace Point-in-Time Recovery

The tablespace moment recuperation (TSPITR) highlight allows you to recuperate at least one tablespaces to a particular moment that is not quite the same as the remainder of the information base. TSPITR is most helpful when you need to:

(i) Recuperate from a wrong drop or shorten table activity

(ii) Recuperate a table that has become legitimately defiled

(iii) Recuperate from an inaccurate clump work or other DML proclamation that has impacted just a subset of the information base

(iv) Recuperate one autonomous construction to a point unique in relation to the remainder of an actual information base (in situations where there are various free patterns in isolated tablespaces of one actual data set)

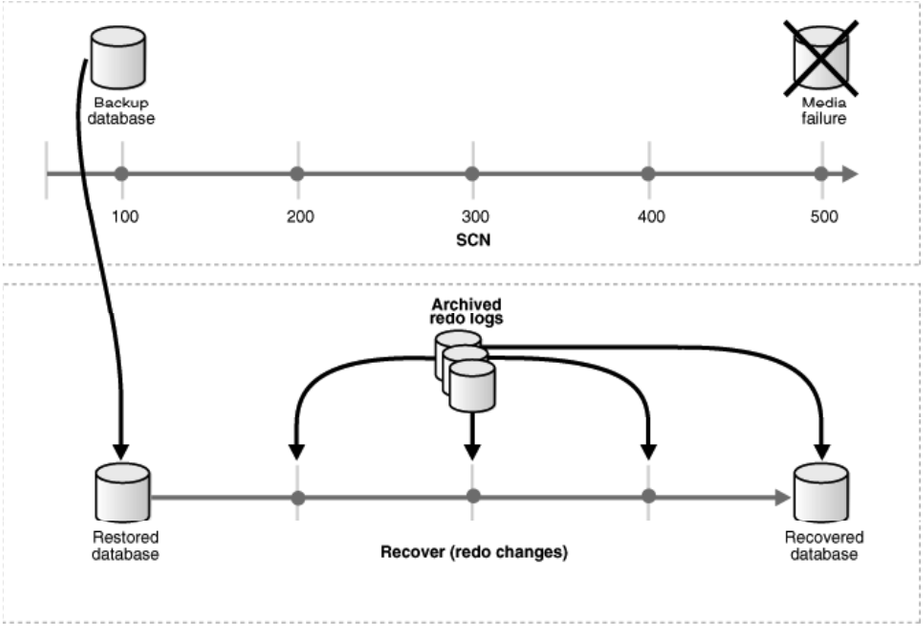


Fig 5.1 *major rule of help up, restoring, and performing media recovery on a data base*

Check Your Progress

5. Define the term JDBC API.
6. What is Oracle information base framework?
7. What is matching?
8. What do you mean the rollback segment?

5.8 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Complex organized information is complicated in nature and is appropriate for the item social highlights of the Oracle data set like assortments, references, and client characterized types.
2. A datatype is assigned to each value limited by Oracle Database. The datatype of a value enables an appropriate sequence of characteristics to be applied to the item. Because of these characteristics, Oracle treats potential gains of one datatype differently than possible gains of another. Potential benefits of the NUMBER datatype, for example, can be added, but not those of the RAW datatype.
3. The value of CHUNK doesn't significantly impact LOBs that are taken care of inline. This happens when ENABLE STORAGE IN ROW is set, and the size of the LOB locater and the LOB data isn't actually around 4000 bytes. Regardless, when the LOB data is taken care of misguided, it for the most part consumes room in results of the CHUNK limit.

4. Web administrations enable application-to-application communication over the Internet, regardless of platform, language, or data design.
5. The JDBC API allows you to call up data sets and SQL orders using Java programming techniques. When you need to get to the database, you can use the JDBC API in a servlet, a JSP innovation page, or a venture bean.
6. Prophet gives programming to make and deal with the Oracle information base. The data set comprises of physical and sensible designs in which framework, client, and control data is put away. The product that deals with the data set is known as the Oracle data set server. On the whole, the product that runs prophet and the actual data set are known as the Oracle information base framework.
7. A query can be created in SQL Server to perform pattern matching using the wildcard characters. The usage of wildcards permits you to find data that matches a certain pattern before specifying it accurately.
8. Rollback segment is actually similar to some other table partitions and record pieces, which involve degrees, moreover demand space and they get made in a tablespace. To play out any DML action against a table which is in a non-structure tablespace ('emp' in 'customer' tablespace), prophet requires a rollback segment from a non-system tablespace.

5.9 SUMMARY

- Straightforward organized information can be coordinated into straightforward tables that are organized dependent on business rules.
- Complex organized information is complicated in nature and is appropriate for the item social highlights of the Oracle data set like assortments, references, and client characterized types.
- Semi-organized information has a sensible construction that isn't regularly deciphered by the data set. For instance, a XML report that is handled by your application or an outer help, can be considered as semi-organized information.
- A datatype is assigned to each value limited by Oracle Database. The datatype of a value enables an appropriate sequence of characteristics to be applied to the item. Because of these characteristics, Oracle treats potential gains of one datatype differently than possible gains of another. Potential benefits of the NUMBER datatype, for example, can be added, but not those of the RAW datatype.
- The value of CHUNK doesn't significantly impact LOBs that are taken care of inline. This happens when ENABLE STORAGE IN ROW is set, and the size of the LOB locater and the LOB data isn't actually around 4000 bytes. Regardless, when the LOB data is taken care of misguided, it for the most part consumes room in results of the CHUNK limit.
- Web administrations enable application-to-application communication over the Internet, regardless of platform, language, or data design.

- Prophet gives programming to make and deal with the Oracle information base. The data set comprises of physical and sensible designs in which framework, client, and control data is put away. The product that deals with the data set is known as the Oracle data set server. On the whole, the product that runs prophet and the actual data set are known as the Oracle information base framework.
- The Database Configuration Assistant makes an information base from layouts that are provided by Oracle, or you can make your own. It empowers you to duplicate a preconfigured seed data set, consequently saving the time and exertion of creating and altering an information base without any preparation.
- After browsing the SQL Server you can search all the columns, rows or records of all the tables in a given database or any specific table for a specific keyword.
- A query can be created in SQL Server to perform pattern matching using the wildcard characters. The usage of wildcards permits you to find data that matches a certain pattern before specifying it accurately.
- Rollback segment is actually similar to some other table partitions and record pieces, which involve degrees, moreover demand space and they get made in a tablespace. To play out any DML action against a table which is in a non-structure tablespace ('emp' in 'customer' tablespace), prophet requires a rollback segment from a non-system tablespace.

5.10 KEY TERMS

- **Straightforward organized information:** It is coordinated into straightforward tables that are organized dependent on business rules.
- **Complex organized information:** It is complicated in nature and is appropriate for the item social highlights of the Oracle data set like assortments, references, and client characterized types.
- **Java persistence API:** It is a diligence solution based on Java innovation concepts.

5.11 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is unstructured information?
2. State about the Consume Datatype?
3. What is the role of SQL?
4. What do you mean Java naming and directory interface?
5. What is checking and tuning execution?

6. How is a database created?
7. What is Backup and Recovery?

Long-Answer Questions

1. Explain the large objects by giving appropriate example.
2. Discuss the available datatypes.
3. Explain the controlling and selecting LOB values by giving appropriate examples.
4. Describe the specifying storage for LOB data with the help of appropriate example.
5. Discuss web enabled database with appropriate examples.
6. Explain the role of Java and WebDB along with its applications.
7. Describe the database administration by giving appropriate examples.
8. Explain the creating and managing rollback segments.

NOTES

5.12 FURTHER READING

Snowdon. 1998. *Oracle Programming With Visual Basic.* India: John Wiley & Sons.

Ying Bai. 2021. *Oracle Database Programming with Visual Basic.*NET. India: Wiley-IEEE Press. First Edition.

Byrla. 2017. *Oracle Database 12C*. India: McGraw Hill Education. First Edition.

P.S Deshpande. 2011. *SQL & PL/ SQL for Oracle 11g.* India: Dreamtech Press.

NOTES

NOTES

NOTES