2023

Relational Database Management System

Dr. Babasaheb Ambedkar Open University

Relational Database Management System

Course Writer

Dr. Amit Bardhan	Assistant Professor, Computer Science Department, Som Lalit Education and Research Foundation, Ahmedabad
Dr. Badal Kothari	Assistant Professor, Department of Computer Science, Hemchandracharya North Gujarat University, Patan
Dr. Vinod Desai	Assistant Professor, Gujarat Vidhyapith, Ahmedabad

Content Reviewer

Prof. (Dr.) Amit Ganatra	Dean, Faculty of Technology and Engineering School of Computer Science, Charotar University of Science and Technology, Changa

Content Editor

Prof. (Dr.) Nilesh K. Modi	Professor & Director, School of Computer Science, Dr. Babasaheb Ambedkar Open University

ISBN-

Printed and published by: Dr. Babasaheb Ambedkar Open University, Ahmedabad

Dr. Babasaheb Ambedkar Open University

MCA-102

Relational Database Management System

Block-1: Fundamental of Database Management System

Block-2: Relational Data Model and Introduction to Oracle Server

Block-3: Oracle Server and SQL

Block-4: Introduction to PL/SQL

Block-1

Fundamental of Database Management System

Unit 1: Basic Concepts of DBMS

1

Unit Structure

1.1. Learning Objectives

1.2. Introduction

1.3. Basic Concepts

1.4. Data

1.5. Database

1.6. Database systems

1.7. Database management system

1.8. Purpose and advantages of database management system

1.9. DBMS Functions

1.10. Disadvantages of database systems

1.11. Check Your Progress

1.1 LEARNING OBJECTIVE

By the end of this unit you should be able to:

- Differentiate between data and information.
- Understand the importance of database and DBMS.

1.2 INTRODUCTION

In today's competitive environment data and its proficient administration is the most significant business objective of any firm. The fact is we are in an era where people are bombarded with huge amount of information explosion. Due to this it becomes difficult to fetch correct information at right time to make decisions properly. Therefore success of every business is highly dependent on how the data is collected, stored and processed for timely decision making.

Any information system like online shopping websites, inventory management systems, clinic management software, online trading applications etc. needs database to store and retrieve the data at regular intervals. DBMS acts as backend for all the different web based and desktop based applications. We cannot imagine a single sector were DBMS is not being used. For example banking, e-governance, logistics, universities, airlines agencies, ticket booking, accounting & filing and every other kind of human endeavor. The management of data in all these systems is done by the means of a general purpose software package called a database management system.

A database management system is a tool to manage the data and perform various activities that include:

- ✓ Creating different databases.
- ✓ Craft required table structures.
- ✓ Inserting records in the tables.
- ✓ Retrieving information from the different tables based on criteria.
- ✓ Deleting the records based on various conditions.
- ✓ Updating the records wherever and whenever necessary.
- ✓ Changing the table structure if required. Etc.

Apart from the above mentioned basic functionalities of the database management system, there are plenty of other functionality like creating users and assigning roles to them, security management, transaction management, managing system catalog, data dictionary management, data backup and recovery etc which are being managed by DBMS.

The role of the DBMS is to act as an intermediary between the users and the database. The DBMS interprets and processes client's requests to fetch the required information from a database. It serves as an interface in several forms like it can be directly accessed from a terminal or using some high level language programs for individual or batch data processing. The request from DBMS to perform various actions is given in terms of SQL (Structure Query Language), which you will be learning in the upcoming units. DBMS shields the database users from the complexity of tedious programming they would have to do to organize data for storage, or to gain access to it once it was stored. Here are going to learn about Relational Database Management System (RDBMS) that stores data in the form of associated tables. Most common examples of RDMS include MySQL, Oracle, PostgreSQL, Microsoft SQL Server etc.

1.3 BASIC CONCEPTS

Storing data, processing it as per requirement and retrieving the required information has been a necessity in each and every organization today. The term data can be explained in terms of "A set of isolated an unrelated raw fact with an implicit meaning". In simpler terms data is a raw fact. It can be anything such as a name of a person, designation of an employee, an audio, video, designation of a person etc. After performing a series of action on the data what we get is an meaningful information. Thus information can defined as data with some fixed and definite meaning. For example, "The cost of the book for programming in python is 750 Rs" is an example of information.

Generally data is what goes into a data processing system and information is the processed data that comes out of the data processing unit.

Limitations of the File based Systems:

- Separation and isolation of data
- Duplication of data
- Structural and data dependence
- Extreme programming effort
- Cannot execute ad hoc queries
- Security features are likely to be insufficient
- System management is complex and complicated

1.4 DATA

Data is nothing but a *raw fact* from which information is generated. Data alone does not have any meaning unless it is organized or arranged in some logical manner. A user must ensure that only valid and significant data must go into the system else the information obtained may not be that trustworthy for the purpose of decision making.

The smallest piece of data that a computer understands is a single character, for an example letter 'S', or a number '6' or a special character '$'. A single character requires one byte of storage.

A character or a group of character that has some specific meaning is called a *field*. A field name uniquely identifies each field.

A logically related set of one or more fields that describe an entity or real world object is called a *record*. For example the fields that constitute bank account record are account number, name, address, pincode, account type, opening date, mode of operation etc.

A collection of related records is called a *table*. An example of department table is given below:

Dept_no	Dept_name	Location
10	Finance	Ahmedabad
20	Purchase	Rajkot
30	Marketing	Bhavnagar
40	EDP	Baroda

Records

Fields

Figure 1: Department Table

1.5 DATABASE

A database is a collection of well organized data in the computer's storage systems that can be used by the application software for some given enterprise. The stored data can be accessed, processed and presented by DBMS to serve a specific purpose. The term enterprise can thought in terms of any individual or large body like a university, bank, logistics company, warehouse etc.

In general database is a shared, collective system construction that stores a collection of:

- End user data. i.e. the raw facts
- Metadata or data about the data.

Here the metadata provides a detailed explanation of the data, its distinctiveness and set of associations or relationships that links the data. Given the uniqueness of metadata, database can be described as a "collection of self-describing data."

1.6 DATABASE SYSTEMS

A database system is principally an automated record maintenance system whose overall reason is to store information and to permit the users to manipulate the information as per requirement. Here we are using the term *data* to refer to what is in point of fact stored in the database and *information* to refer to the meaning of data as understood by the client.

Database system is obtainable on all the machines that range from the smallest handheld devices to PC's to large main frame computers.

1.7 DATABASE MANAGEMENT SYSTEM

A database management system (DBMS) is a compilation of programs that manages the database structure and controls access to the data stored in the database. DBMS serves as a mediator between the client and database by hiding all the complexities from the end user.

1.8PURPOSE AND ADVANTAGES OF DATABASE MANAGEMENT SYSTEM

The DBMS receives the entire applications request and translates them into the complex operations that are required to fulfill those requests. It also hides the internal complexity from the application programs and users. The applications programs can be written in any language like Python, Java, C++ etc.

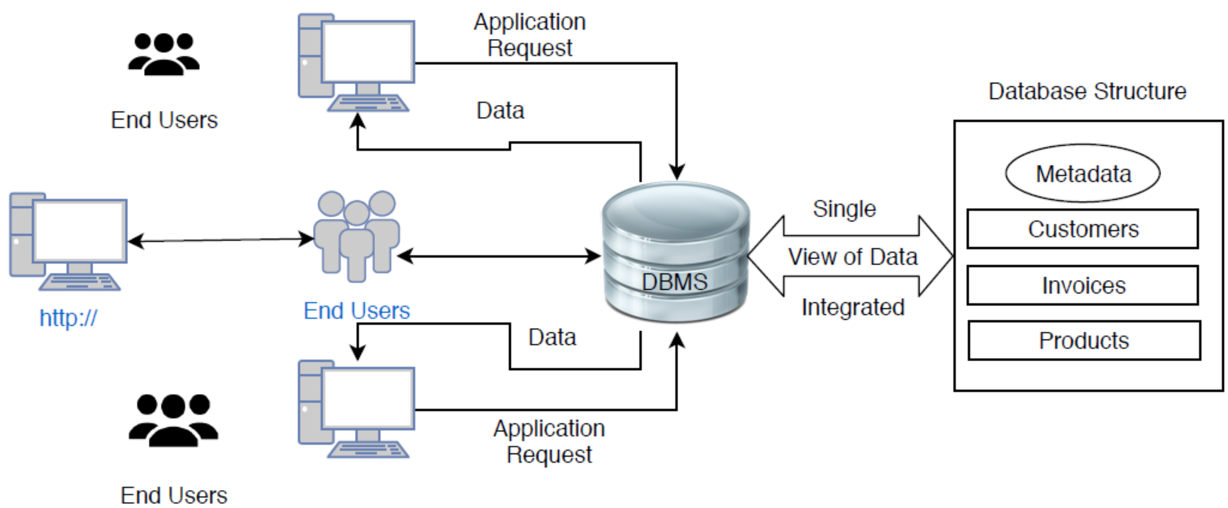


Figure-2 DBMS managing the functions between the client and the database

DBMS also allows the data to be shared among the multiple applications or clients and helps in merging many different views of data into single data repository. In particular DBMS provides the following advantages over the files system:

- *Better data sharing capabilities:* The DBMS helps to generate an environment in which the end users locally or globally can have access to the data for quick decision making.

- *Enhanced data security:* DBMS provides a structure to implement data privacy and security policies. Different categories of roles can be created for special users and rights can be given accordingly.

- *Superior data integration facilities:* Wider admittance to well managed data promotes an incorporated view of the organizations operations and a apparent view of the complex picture.

- *Reduced data inconsistency:* It exists when different versions of same data appear in diverse locations. For example data inconsistency exits when the name in your bank account and the name on your cheque book differ. This possibility can be reduced by properly designing the database.

- *Faster data access:* When dealing with huge amount of data DBMS makes it possible to produce quick answers to any queries by using SQL. Example queries can be how people have deposited notes of 500 denominations at the time demonetization in ABC branch.

- *Improved decision making:* If the data is managed properly and faster data access is done it makes probable to produce enhanced superiority information, based on which better decisions can be taken.

- *Improved end user productivity:* The ease of use of data, shared with the tools that alter data into usable information, allow end users to make rapid, knowledgeable decisions.

1.9 DBMS FUNCTIONS

A DBMS performs quite a lot of significant functions that promises the reliability and uniformity of the data in the database. Few of the important functions are mentioned below:

- ✓ *Data transformation and presentation:* The DBMS converts the entered data to confirm with the required data structures; therefore it relieves you from the task of making distinction between logical and the physical format. For example the date the format in INDIA is DD/MM/YYYY, but in MySQL is YYYY-MM-DD, so transformation in to the required format can be easily made.

- ✓ *Multiuser access control:* To provide data steadiness DBMS uses classy algorithms to make sure that multiple users can access the database in parallel without compromising the integrity of the database.

- ✓ *Security Management:* DBMS enforces user security at different levels in order to provide which data operations a group of users or a particular user can perform. DBMS assigns access privileges for various database components.

- ✓ *Data dictionary management:* DBMS stores definitions of data elements and their metadata. It uses data dictionary to come across up the necessary data constituent structures and its associations.

- ✓ *Data storage management:* A modern DBMS provides storage not only for the facts but also for associated data entry forms, report definitions, data validation regulations, formations to handle audio and video formats and so on. It actually stores the database in multiple physical data files.

- ✓ *Backup and recovery management:* To provide data safety and integrity DBMS provides backup and recovery control. It basically deals with the recovery of bad sector in the disk and also data recovery at the time power failures.

- ✓ *Data integrity management:* DBMS supports and implement integrity regulations, thus minimizing data repetition and increasing consistency.

- ✓ *Database access languages and API:* DBMS make available data access through a query language called SQL. Structured Query Language (SQL) is a de facto query language supported by majority of the DBMS vendors. Apart from that DBMS also provides application programming interfaces to main programming languages like Python, C#, Java, Magento, PHP etc.

- ✓ *Database communication interface:* DBMS provides admittance to the database via command line terminals, via web browsers (GUI) etc.

1.10 Disadvantages of Database System

DBMS do carry significant disadvantages as mentioned below:

- *Increased cost:* Database system needs sophisticated hardware and software and extremely capable expert to manage it. Thus the cost of managing the people, software and hardware and providing training, licensing add an extra overhead to cost.

- *Management Complexity:* Database system boundary with many diverse technologies and are can become more and more complex in order to handle day to day transactions.
- *Maintaining currency:* To make the most of the database it is required to keep your systems current. That leads to frequent upgrades and increased in training cost.

- *Vendor Dependence:* The end users are heavily vendor dependent since they are storing each and every information into the database. On the contrary the vendors are less likely to offer pricing point reward to the existing clients.

Frequent Upgrade cycle: DBMS vendor repeatedly advance their products by incrementing new functionalities. And many a times those software upgrades requires new hardware resources.

1.11 Check your progress

1. Define the following terms:
   a. Data

b. Information

c. Field

d. Record

2. List and explain the limitations of file based systems.
3. Discuss the purpose and advantages if DBMS.
4. List and explain DBMS functions in detail.
5. Explain the potential cost of implementing a database system.

Unit 2: Architecture Of DBMS

2

Unit Structure

2.1 LEARNING OBJECTIVE

By the end of this unit you should be able to:

- Understand the basic architecture
- Understand basics components of DBMS

2.2 INTRODUCTION

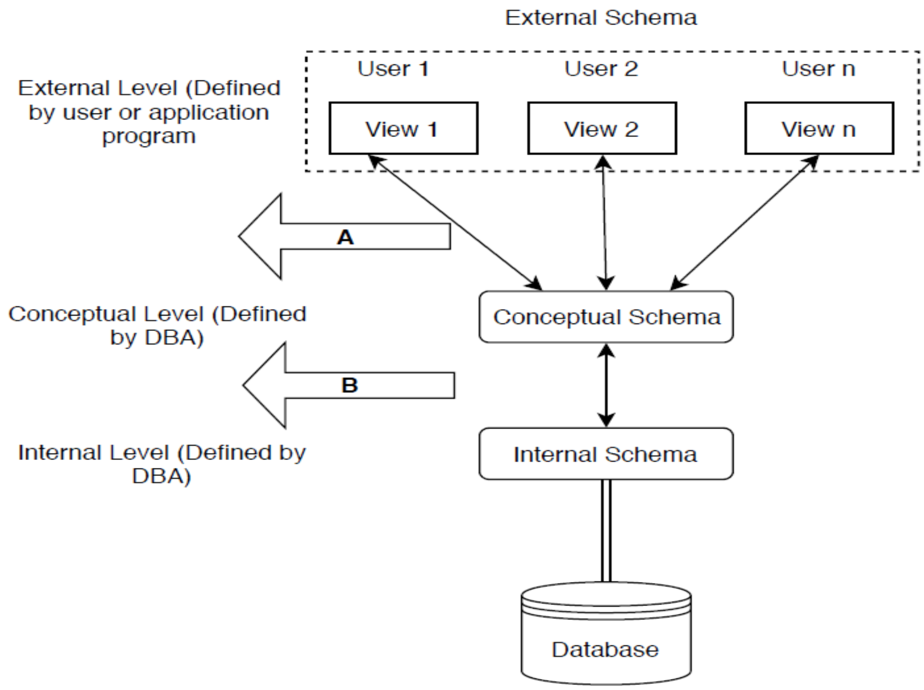
DBMS is very sophisticated software application that provides reliable management of large amounts of data. To understand all-purpose database concepts and the structure and capabilities of a DBMS better, the structural design of a typical DBMS must be known.

2.3 ARCHITECTURE OF DBMS

The DBMS architecture describes how the data in the database is viewed by the different users. This architecture provides the data at different levels of the abstraction to the users by hiding the complexities of its internal management activities.
In this architecture the overall database description can be defined at three levels:

- Internal
- Conceptual
- External levels

For this reason many a times it's known as three-level DBMS architecture. The architecture is proposed by ANSI/SPARC (American National Standard Institute/ Standards Planning and Requirement committee).



A-External / Conceptual Mapping (Logical Data Independence)
B-Conceptual / Internal Mapping (Physical Data Independence)

Figure-3 Three Level DBMS Architecture

External Level:

It is the highest level of abstraction that deals with the user's view of database and therefore it's also known as view level. The external level describes the part of the database to a specific group of users or to an individual user.

Each view available to the user is customized to their requirements. It may be possible that same data may be visible to different users through different interfaces. In this way it also provides a powerful and flexible security mechanism by hiding certain data from certain users. The data described at this level is independent of both hardware and software. Generally entity relationship diagram is used to represent the external view as the data is modeled.

Conceptual Level:

This level of abstraction deals with logical structure of the entire database and is also known as logical view. The view describes the structure and the type of the data that is stored in the database along with the relationships among the data.

It describes all the requirements of the users without the description of physical implementation. It is the overall view of the database keeping in the consideration the

DBMS software that is going to be used. This view is thus dependent on the software but independent of the hardware.

Internal Level:

This level describes data at the lowest level of abstraction that deals with physical representation of the database on the computer and is also known as physical level. It describes how the data is stored and is organized on the physical storage medium.

At this level various aspects are considered to achieve optimal runtime performance and storage space utilization. This level is dependent on the software (mostly the OS) as well as hardware.

To understand the three-level database architecture consider the example of Employee database as shown in the figure 1.4. In this figure two views (View 1 and View 2) of the Employee database are defined at an external level. Hence different users can see different external views that they queried. The details about the data type and the size of the fields are hidden from the users at the external level.

At the conceptual level the employee records are described along with their data types. The application programmers and the DBA generally work at this level of abstraction. At the internal level the employee records are described as a block of consecutive locations such as words or bytes. The database users and the applications programmers are not aware of these details; however the DBA may be aware of certain details of the physical organization of the data.

When a user specifies a request to generate a new external view, the DBMS must transform the request specified at the external level into a request at conceptual level followed into a request at physical level. If the user requests for data retrieval, the data extracted from the database must be presented according to the need of the user. This process of transforming the requests and results between various levels of DBMS architecture is known as mapping.

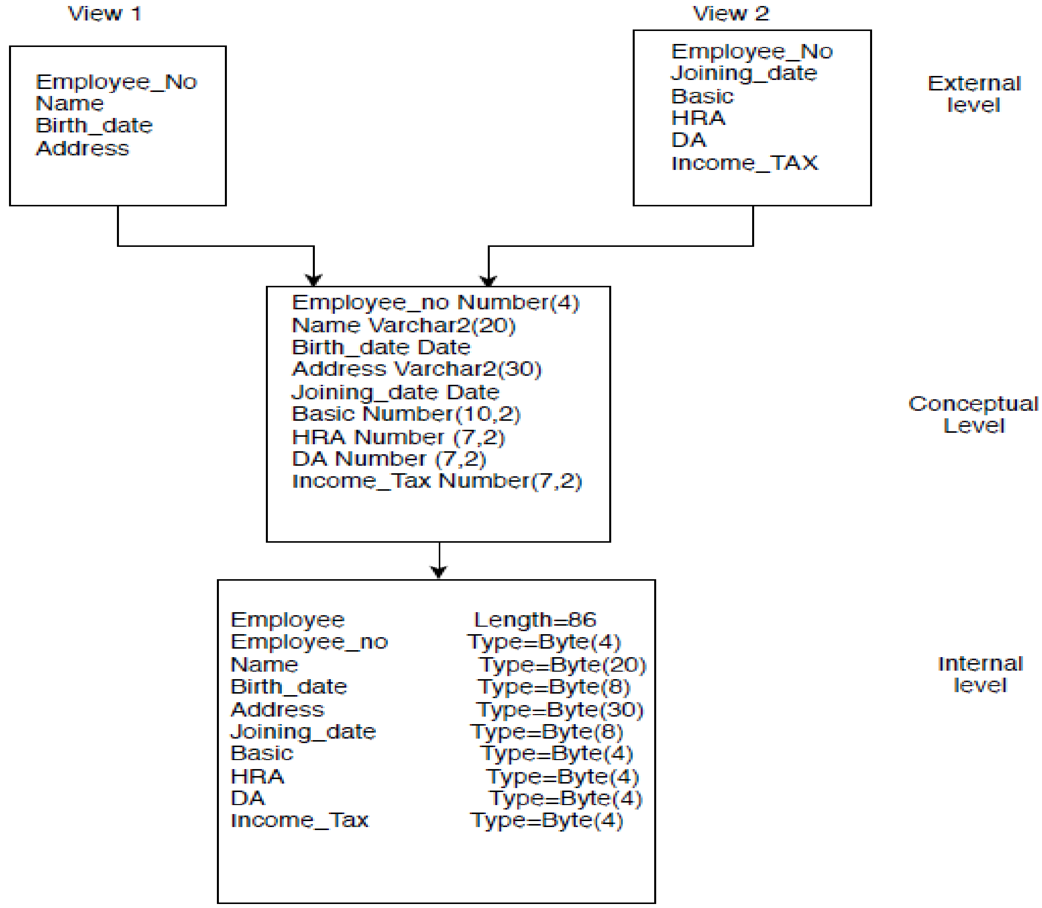


Figure-4 Three levels of Employee Database

The main merit of three-level DBMS architecture is that it provides data independence. Data independence is the ability to change the schema at one level of the database system without changing the schema at the other levels. Data independence is of two types:

Logical Data Independence:

The ability to adapt the conceptual level without altering the external level or application program is known as logical data independence. The conceptual schema can be changed due to the change in constraints or addition of new features. This change will have no effect on the external level schema that is already there. Logical data independence is difficult to achieve as the application programs are always dependent on the logical structure of the database. Therefore changes in the logical structure of the database may require change in the application program.

Physical Data Independence:

The ability to change the internal level without changing the conceptual level is known as ph ysical dat a i ndependence. T he t ransform i n t he da ta s torage s tructure or ac cess strategy or i ndexing t echnique w ill hav e no ef fect on t he c onceptual s chema. T his i s because the mapping between the conceptual schema and the internal level is provided mostly by DBMS and changes are taken care of by mapping. Hence the physical data independence is easy to achieve.

2.4 VARIOUS COMPONENTS OF DBMS

The database system is composed of five major components, that is:

- Hardware
- Software
- People
- Procedures and
- Data

Let's take an individual look at the five components:

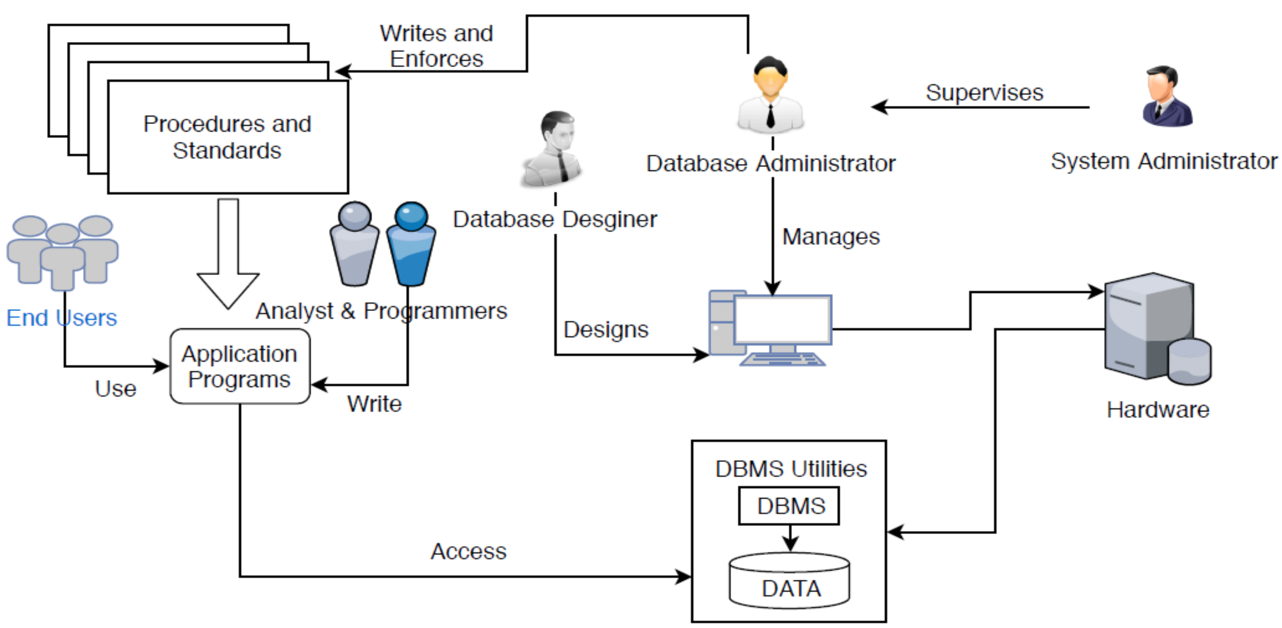


Figure-5 Database system environment

- ***Hardware*:** It refers to all the system's physical devices that can be storage devices, network devices, printers, servers, workstations, computer etc. The computer may range from personal computers to a main frame and it may include one powerful server depending upon the organizations requirements and the size of the database.

  A good database system requires a database server with a fast processor and significantly large amount of main memory. It also includes different kind of peripheral devices to handle various kinds of data. The advancement in computer hardware technology and development of powerful computers has resulted into increased database technology development and its application.
- ***Software:*** There are basically three types of software needed:
  - *Operating System*: It manages all the hardware components and makes it possible for the software to run on the computer. Most commonly used operating systems are LINUX, WINDOWS, MAC etc.
  - *DBMS:* DBMS software manages the data in the database. Some examples of commonly used DBMS software include- MySQL, Oracle, DB2, MSAccess etc.
  - *Application programs and utility software:* It is used to access and manipulate data in the DBMS. Applications programs are used to provide an interface to accept data from the user. They are also used to access data from the database in order to provide reports, tabulations and other logical information to the user. Utility software is used to help manage some DBMS components.
- ***People:*** It includes all the users who interact with any component of the database system environment. List of all the users are listed below:
  - *Database Administrator:* DBA is one of the main user responsible for managing the DBMS and controlling the database of the DBMS. DBA is mainly responsible for setting up procedures and standards and ensuring that they are implemented properly.

- *System Administrator:* System administrator is the one who takes care of all the computers in the network, and the database systems general operations.

- *Database Designer:* They are also called data base architects. They along with the database administrator design the structure of the database. If the database design is poor other all components of the database system environment become helpless.

- *System Analyst and Programmers:* They design and implement the application programs. They are responsible for designing the forms and reports. They may also set up procedures through which end users access and manipulate the data in the database.

- *End User:* They are those users who use the application programs to manage the day-to-day operations of the business. End users include all employees of the organization starting from the data entry operators to the decision makers. Some of them enter raw data and some of them process the raw data and generate information.

- ***Procedure:***Procedures are instructionsand rules that govern the design and use of the database system. Procedures help to maintain certain level of standards and ensure that the data entering the system and information generated from the system are all in well organized manner.

- ***Data:***Data is nothing but raw facts from which the information is generated. Data actually includes the entire collection of data that goes into the database. Only valid and significant data must go into the system else the information obtained may not be reliable for the purpose of decision making.

2.5 Check your progress

1. Explain the 3-level database architecture in detail.
2. What is data independence? Explain in brief logical data independence and physical data independence.
3. Write a short note on database system environment.

Unit 3: Data Models

3

Unit Structure

3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Evaluate different data model and its mapping.

3.2 INTRODUCTION

One of the main objectives of the database systems is data abstraction that is to highlight only the essential features and to hide the storage and data organization details from the user. A model is an abstraction process that concentrates on essential and intrinsic features of the application while ignoring the details that are not important. A database model provides the necessary means to achieve data abstraction. A data model allows the conceptualization of the association between entities and its attributes.

A data model is a simple demonstration, generally graphical, of more complex real word data structures. It consists of a set of data structures and conceptual tools that is used to describe the structure (Data types, relationships and constraints) of a database.
A data model not only describes the arrangement of the data, it also defines a set of operations that can be performed on the data. A data model generally consists of data model theory, which is a formal description of how data may be structured and used, and data model instance, which is practical data model designed for a particular application. The process of applying data model theory to create a data model instance is known as data modeling.

3.3 DATA MODELING

A data model can be very useful communication tool that provides a means of interaction between the databases designer, application programmer and the end user. There are different types of data model that are explained in the next section.

3.4 THE HIERARCHICAL DATA MODEL

The hierarchical model was the first proper model developed. Its basic logical structure is represented by an upside down tree.

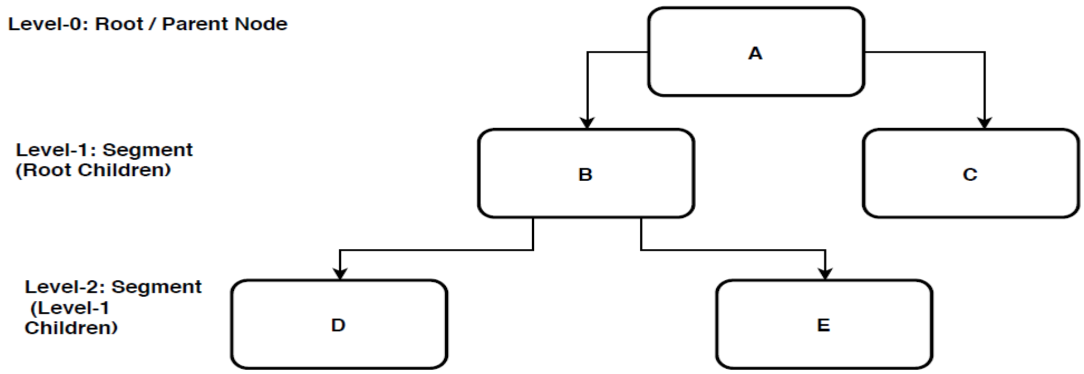


Figure-6 Hierarchical Data Model

The hierarchical structure contains levels or segments. A segment is equivalent to a file system record type. With the hierarchy the top most level or segment is known as a root node or the parent node. The root node or the parent node is assigned the level – 0 as shown in the Figure-6. Again within the hierarchy each segment is perceived as a parent of the segment below it.

In other words, each record is perceived as a parent record of the segment or the child record below it. As shown in the Figure-6 the segment at level-0 i.e. the root node is the parent node for the segments at level-1. Similarly the records at level-1 are also parent records for those records at level-2.

The hierarchical data model is best suitable to represent one-to-many relationship as shown in figure 1.6. In this model each parent record can have multiple child records related to it. The limitation of this model is one child record can have only one parent record. Hence it is difficult to represent many-to-many relationship using this model.

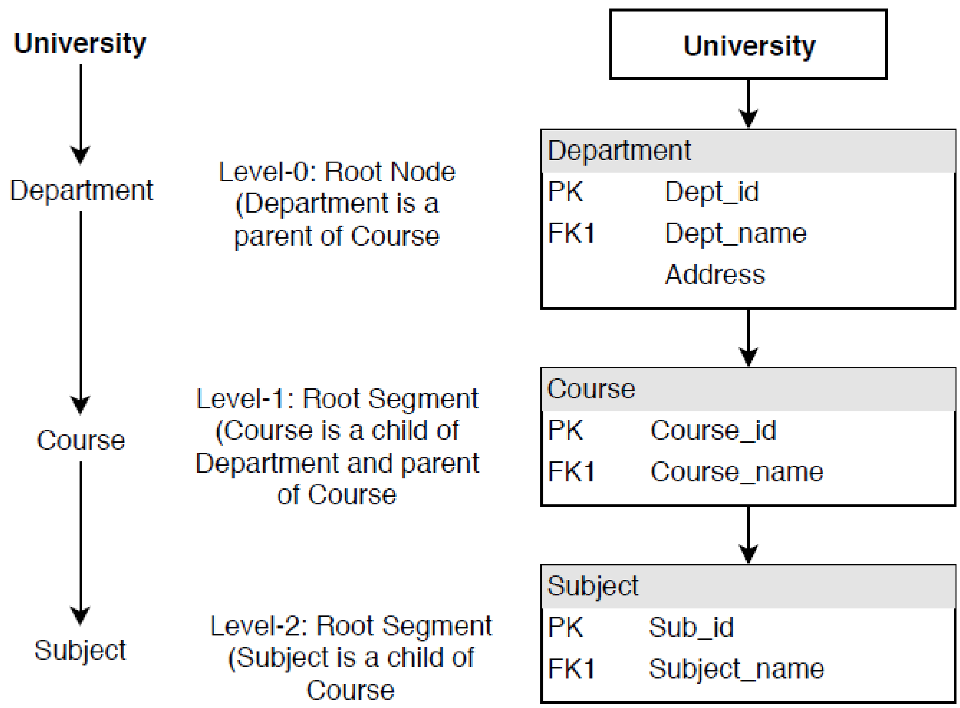


Figure-7 Hierarchical data model relationship

Figure-7 shows a hierarchical data model of a university tree type consisting of three levels. A single college record at the root level represents one instance of the department record type. Multiple instances of a given record are used at lower level to show that a department may consist of many courses and one course may consist of many subjects.

Merits of Hierarchical Model:

- ✓ **Simplicity:** It is simple and easy to understand and implement as the relationship between the various layers is logical and always 1:M

- ✓ **Data Integrity:** The parent/child relationship is always there between the layers. The model promotes data integrity as the child segments are automatically referenced to its parent segment.

- ✓ **Efficiency:** It is very efficient when the database contains large amount of data in 1: M relationships and when large number of transaction are required using data, having relationship fixed over time.

- ✓ **Data Sharing:** Data sharing becomes practical as all the data are held in a common place.

Demerits of Hierarchical Model:

- ✓ **Implementation complexity:** It is quiet complex to implement as the DBMS requires the knowledge of physical level of data storage and the database designers should have a very good knowledge of the physical data storage characteristics.

- ✓ **Implementation limitation:** The model does not allow one child record to be related to multiple parent record types. This poses great difficulty in representing many-to-many relationship.

- ✓ **Inflexibility:** The changes in the new relation or segments often yield very complex management task. The deletion of one segment will cause all other segments below it to be deleted.

- ✓ **Database Management problems:** If any changes are made to the database structure, it becomes essential to change all the application programs that access the database.

- ✓ **No standards:** There are no laid down set standards on how to implement the model.

3.5 NETWORK DATA MODEL

The network model was created to represent complex data relationship more effectively then the hierarchical model, to improve database performance, and to impose a database standard.

The network model is similar to the hierarchical data model except that a record can have multiple parents. This model has three basic components such as record type, data items and links.

A relationship is called a set in which each set is composed of at least two record types- owner record (same as parent record) and member record (same as child record). The connection between an owner and a member is identified by a link to which a set name is assigned.

The set name is used to retrieve and manipulate data. The link between the owners and their members indicate access paths in the network model and are typically implemented with pointers. In network data model, member can appear in more than one set and thus may have several owners, and hence it facilitates many-to-many relationship. A set represents a one-to-many relationship between the owner and the member.

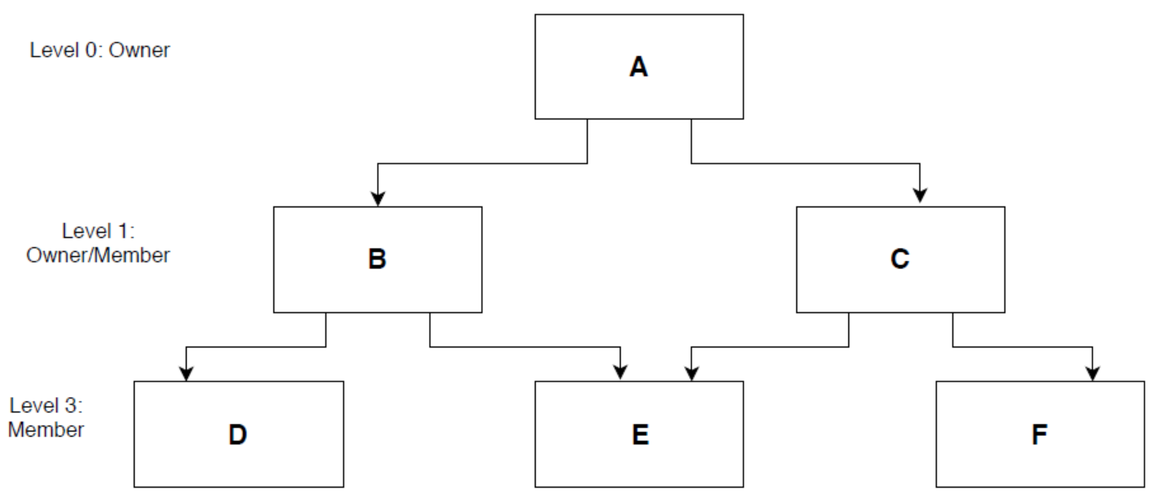


Figure-8 Network Data Model

In the above diagram a sample network data model is represented. As shown the member ‘B’ has only one owner ‘A’, whereas member ‘E’ has two owners ‘B’ and ‘C’.

The figure-9 it demonstrates a distinctive network model representation for sales process. The model represents five record types namely- Sales_person, Customer, Item, Sales_order, Billing and Order_detail. Here the entity Sales_order has two owners Sales_person and Client. Similarly Order_detail has two owners Item and Sales_order. In this model eack link between two record types represents a one-to-many relationship between them.

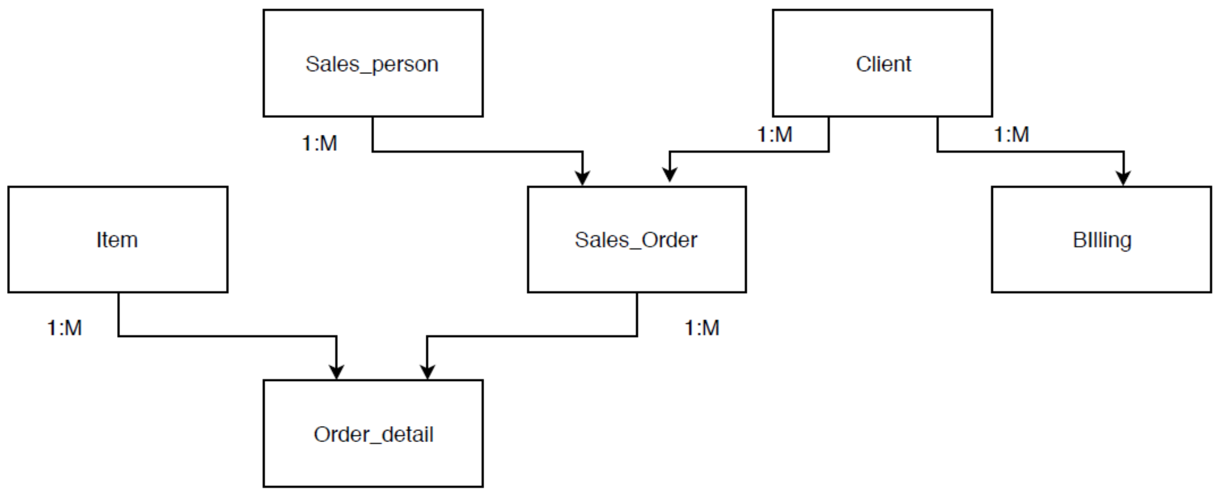


Figure-9 Network Model for Sales Process

Merits of Network Model:

- ✓ **Simplicity:** Same as hierarchical model network model is also simple and easy to understand.

- ✓ **Facilitating more relationship types:** The network m odel i s abl e t o h andle many-to-many relationship as a member can have multiple owners. This helps in modeling real life situations in a much better way.

- ✓ **Superior Data Access:** An app lication c an ac cess an ow ner r ecord and al l the member record within the set. Hence the data access and flexibility found in this model are much better as compared to the hierarchical model.

- ✓ **Database Integrity:** It e nforces i ntegrity a nd d oes not allow a m ember to exist without an owner.

- ✓ **Support for DBMS:** It includes Data Definition Language (DDL) and Data Manipulation Language (DML) in DBMS.

- ✓ **Database Standards:** It is based on universal standards formulated by DBTG (Database task group) / CODASYL (Conference on data system languages) and improved by ANSI/SPARC.

Demerits of Network Model:

- ✓ **System Complexity:** Network models are difficult to design and use properly. The navigational access mechanism accesses only one record at a time and hence makes the system implementation very complex. Knowledge of the internal data structure is required to take the advantage of this model.

- ✓ **Absence of Structural Independence:** If changes are made to database structure, all subschema descriptions have to be updated before any application program can access the data.

3.6 RELATIONAL DATABASE MODEL

The relational data model was originally commenced by Dr. E.F. Codd. It is implemented through a very sophisticated relational database management system (RDBMS). IT not only performs the same basic functions that are there in hierarchical and network model but also provides the ability to hide the complexities of the relational model from the end user. Table is a matrix consisting of series row/column intersections related to each other through sharing a common entity characteristic. Relational diagram is a representation of relational database's entities, attributes within those entities, and relationship between those entities. Relational table stores a collection of related entities and resembles a file. Relational table is purely a logical structure and how data are physically stored in the database is of no concern to the user or the designer.

In relational data model, tables are related to each other through the sharing of common attribute. For example the Subject table in the given Figure 1.10 contains Faculty_id field and the same filed also exists in the Faculty table.

Faculty

Faculty_id	Faculty_name	Department
F001	Dr. Nilesh Modi	Computer Science
F002	Prof. Ramesh Kataria	Mathematics
F003	Prof. Kamesh Raval	Computer Science
F004	Dr. Amit Ganatra	Computer Science

Subject

Subject_id	Subject_name	Faculty_id
S001	Computer Network	F001
S002	Discrete Maths	F002
S003	Database	F004
S004	.Net	F003

Figure-10 Relational Data Model

The common field between Faulty and the Subject tables allows a subject to match with the details of the faculty who is teaching it. Here although the tables are independent of each other, the data between the two tables can be easily associated. The relational database provides the least amount of redundancy.

Merits of Relational Data Model:

- ✓ **Conceptual Simplicity:** The tabular view of storing and managing the data improves conceptual simplicity, thereby encouraging easier database blueprint, implementation, administration and usage.

- ✓ **Structural Independence:** The relational data model does not depend on the navigational data access and hence the changes in the table structure do not affect the data access.

- ✓ **Flexible and powerful query capability:** It provides very powerful, flexible and easy to use querying facilities. IT has SQL to execute the required data operations and manipulations.

- ✓ **RDBMS support:** The availability of powerful RDBMS isolates the end user from the physical-level details and improves execution and administration ease.

Demerits of Relational Data Model:

- ✓ **Hardware Overhead:** This model requires a fast processor along with a large capacity and high speed secondary storage devices to perform the assigned tasks. Now-a-days this is not that big disadvantage as the computing speed is getting doubles every eighteen months and the cost of storage devices are getting reduced to a great extent.

- ✓ **Poor Design by untrained professionals:** Because of ease of use many a times it is managed by untrained professional to develop the required queries. So queries and reports written without proper logical thinking results in slower system and performance degradation.

3.7 ENTITY RELATIONSHIP DATA MODEL (ER MODELS)

The Entity relationship model was initially projected by Peter Chen in 1976. It is a graphical representation of database structure using entities and relationship among entities. The ER Model matched the relational data model very satisfactorily. The combination provides a very good database design.

The ER model is has following components mentioned below:

Entity Set: It is a real world object for which data are collected and stored. It is just one instance of an entity set. The term entity and entity set are different but can be used interchangeably. An entity set is represented by a rectangle in an ER diagram. The name of the entity is generally noun and singular. Examples of entity are Department, Course, Student etc.

Attributes: The characteristics of an entity is called an attribute. One entity can have multiple attributes like an entity Course can have Course_id, Course_name, Duration etc are the attributes.

Relationships: It describes an connection between two entities. There are three types of possible relationships between the entities , they are one-to-one (1:1), one-to-many (1:M) and many-to-many(M:N).

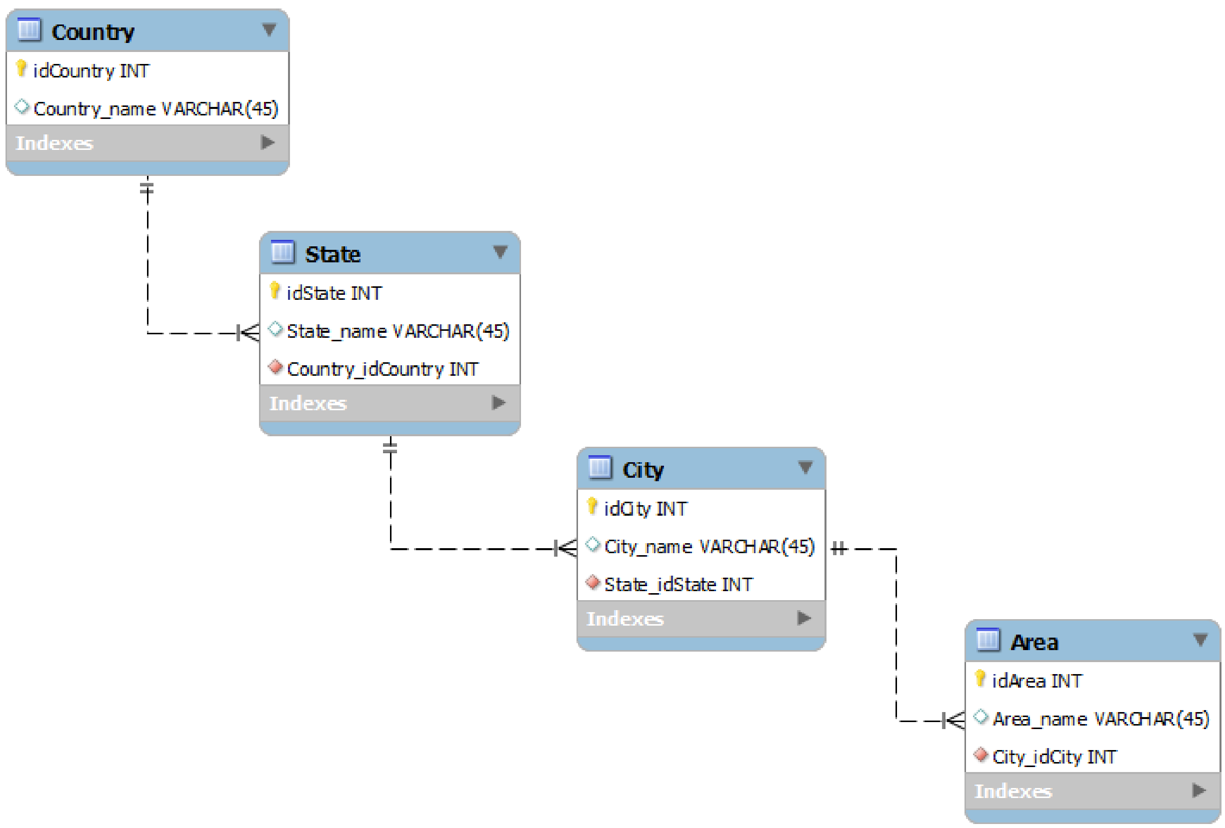


Figure-11 Sample ER Diagram using Crow's Foot Notation

The above figure illustrates 1: M relationship between the entities Country, State, City and Area. The idCountry attribute from the Country table is referenced in State table. Therefore it represents one-to-many (1:M) relationship between the entity Country and State, which means one country can have many states. Similarly there is one-to-many relationship between State entity and City, and 1:M relationship between City and Area entity.

Merits of ER Model:

- ✓ The ER model is a graphical demonstration of entities which results in complete clarity and simplicity in understanding.
- ✓ ER model also goes in combination with the relational model data model and with help of some tools like MySQL Workbench conversion.

Demerit of ER Model:

- ✓ Depending upon different logical perceptions many a times it's not possible to specify most of the constraints.

3.8 OBJECT ORIENTED DATA MODEL (OODM)

The object oriented data model is a logical data model that is based on the concept of object oriented programming. It has come into existence to meet the increasingly complex real world applications which are not being easily solved by other model. A class represents both object attributes as well as the behavior of the entity. The instance of the class- object contains both data as well as their relationship. An object includes information about the relationship between the facts within the object as well as information about relationship with other objects. Objects also contain all operations that can be performed on it.

The object-oriented data model is differently proposed by different researchers and has no single common database structure like the other data models. OODM forms the basis for the object-oriented database management system (OODBMS). They are mainly used in engineering and design, financial services, telecommunications etc. This model is represented by UML (Unified Modeling Language) class diagrams.

The main advantage of OODM is that it is closer to the real word and hence is able to deal with more complex problems very easily. The main demerit of OODM is no established standards and hence is not that much widespread accepted.

3.9 COMPARISON BETWEEN DATA MODEL

We have discussed all the entire data models and based on some specialized characteristics and some merits and demerits we compare all the models. The table given below shows the comparison:

Data Model	Characteristics	Organization	Identify	Access Language	Data Independence	Structural Independence
Hierarchical	Best suitable for 1:M relationship	File, Records	Record based	Procedural	Yes	No
Network	Ability to handle all types of relationship, including M:N	File, Records	Record based	Procedural	Yes	No
Relational	Conceptual Simplicity, easier database design.	Tables	Value based	Non-Procedural	Yes	Yes
Entity Relationship	Visual representation makes it very easy to understand	Entity Sets/ Objects	Value based	Non-Procedural	Yes	Yes

Object Oriented	No standardized method available to represent the model.	Objects	Record based	Procedural	Yes	Yes

Table-1 Comparison between data model

3.10Check your progress

1. Explain the importance data model.
2. Define entity, attributes and relationships.
3. Discuss hierarchical model in detail.
4. Explain in detail the network model.
5. Write a short note on ER model.

Unit 4: Database Design

4

Unit Structure

4.1. Learning Objectives

4.2. Introduction

4.3. Characteristics of a table

4.4. Keys

4.5. Integrity policies

4.6. Relational set operators

4.7. Attributes

4.8. Relationships contained in relational database

4.9. Connectivity and cardinality

4.10. Relationship Strength

4.11. Relationship degree

4.12. Database design process

4.13. Anomalies in database

4.14. Check your progress

4.1 LEARNING OBJECTIVES

After studying this unit student should be able to:

- decide an entity and its attributes.
- understand database design process and the commonly occurred anomalies in it.

4.2 INTRODUCTION

A table is viewed as a two dimensional organization consisting of rows and columns. A table m any a t imes i s al so c alled a *relation* because t he r elational m odel ar chitect composed of r ows and c olumns. A t able c onsists of a c ollection of as sociated ent ity occurrences that is an entity set. For example a DESIGNATION table contains the entity occurrences, each representing a separate designation of an employee.

With the help of table view of data it makes it easy for a database designer to design the database.

4.3 CHARACTERISTICS OF A RELATIONAL TABLE

The eight characteristics of a relational table are mentioned below:

1. A table is perceived as a two-dimensional arrangement structure of rows and columns.
2. Tuple corresponds to a single entity event contained in the entity set.
3. Every relational table column represents an attribute, which should have a distinct name.
4. The intersection of a row and column represents a single value in the table.
5. Every value in the column must correspond to the same data type and format.
6. Each column can have a definite range of values know as attribute domain.
7. The sequence of rows and columns is irrelevant in DBMS.
8. Every table must have an attribute or its combination that can distinctively identify a tuple.

Table: DESIGNATION

Desig_id	Desig_name
1	CEO
2	Manager
3	Supervisor
4	Technician
5	Officer

Desig_id= Designation ID, Desig_name= Designation name

Table-2 DESIGNATION table attribute values

i. The DESIGNATION table is viewed as a two dimensional arrangement consisting of two columns and five rows.

ii. Each row in the DESIGNATION table illustrates a single entity occurrence within the entity set. For example as shown in the figure-12 here any Desig_id=4, represents the other characteristics that's designation name in the given table, the designation name in this case is Technician which denotes a record.

iii. Here each column is viewed as an attribute and should have unique name.

iv. As shown in the given figure the entire attribute in a given column must have a same data type. Like designation name field has a data type as character.

v. Here the designation ID has a range of possible values that are between 1 to 5, which is known as range of domain values.

vi. The series of rows and columns is irrelevant in DBMS.

vii. Each table in RDBMS must have a column/attribute which contains set of unique values and that attribute can be assigned as a Primary Key (PK). Assigning a PK attribute to a field, does not allow the field to remain either empty or repeated.

4.4 KEYS

Keys in RDMS are significant as they are used to ensure that each tuple in a table is uniquely identifiable. A key consists of one or more attributes that determine other attributes. For example an Designation ID identifies all the field in the designation table.
A primary key plays an important role in the relational environment, where the key's role is based on the concept of determination. Each table must have a attribute that is unique and is able to identify the unique records of the table.
Similarly the foreign key contains either matching values (primary key of another table) or nulls. The table that makes use of that foreign key is said to exhibit *referential integrity*. In simple words referential integrity means that if the foreign key contains a value, that value should refer to an existing valid record in another relation.
In the context of database table, the statement " A determines B ", indicates that if you know the value of attribute A, you can look up into the value of B. For example the knowing the Student_ID in the STUDENT table we are able to look up his/her name, score, mobile number etc. Therefore attributes of the student table can be represented by the statement " Student_ID determines name, score, sem, mobile". This statement can be simply denoted by:
Student_ID -> Name, Score, Sem, Mobile_num
The concept of determination is important as it used in the definition of a central relational database concept known as functional dependency. The functional dependency can be defined most easily this way: "The attribute A determines B if all the rows in the table that agree in the value for attribute A also agree in value for attribute B".
Also " If an attribute B is functionally dependent on a composite key A but not on any subset of that composite key, the attribute B is fully functionally dependent on A".
Composite key is a combination of 2 more attributes that is used to uniquely identify a record in a given table. Within the broad key classification special keys can be defined as given the figure 1.14

Key Type	**Definition**
Super key	An attribute that uniquely identifies each row in a table

Candidate key	A minimal (irreducible) superkey.
Primary key	A candidate key that is selected to uniquely identify all other attributes in a column and does not contain a null value
Secondary key	An attribute used strictly for data retrieval purposes.
Foreign key	An attribute in one table whose value must match the primary key in another table.

Table-3 Relational Database Keys

4.5 INTEGRITY POLICIES

For a good relational database design integrity rules are very significant and they must be followed. Several RDMS implement integrity rules without human intervention but care should be taken that any application design must match the referential integrity rules which are summarized in the figure 1.15:

Entity Integrity	**Description**
Requirement Purpose	All primary keys are unique and cannot be null Each row will have a unique identity and the foreign key can reference primary key values. E.g. No Student ID can be duplicated as well as it cannot be null.
Referential Integrity	**Description**
Requirement Purpose	A foreign must match with the primary key value in a table to which it is related, or sometimes may have a null entry. It may be possible for an attribute NOT to have a corresponding value, but an invalid entry is not possible. E.g. An AGENT has yet not assigned any CUSTOMER.

Table-4 Integrity rules

As shown in the Table-5, the STUDENT table does not contain a repeated Student_ID as well as does not contain null which represents entity integrity.

Student_ID	**Name**	**Sem**	**Score**	**MOB**
S001	Amit	I	75	9898989898
S002	Neha	II	83	9090909090
S003	Hem	I	87	7878568923

Table-5Sample STUDENT table

Similarly the tables AGENT and CUSTOMER are shown in the Table-6, where the agent Ramesh and Joy has yet not assigned any customer, and Agent_ID attribute in the Customer table is null for the customer named sumit and harsh.

Agent_ID	**A_name**	**MOB**
1	Nilanshu	7539518526
2	Shyam	4567891236
3	Ramesh	3216549875
4	Joy	3578529631

Customer_ID	**C_Fname**	**C_Lname**	**City**	**Agent_id**
1	Sumit	Verma	Ahmedabad	
2	Nancy	Joseph	Surat	1
3	Jenny	Shah	Rajkot	2
4	Harsh	Modi	Surat	

Table-6 Sample AGENT and CUSTOMER table

4.6 RELATIONAL SET OPERATORS

The data in the RDBMS are of limited worth until we can manipulate to generate useful information. In this section we will be describing about eight relational set operators populated by relational algebra to implement various operations. The operators that we are going to discuss are: UNION, INTERSECT, DIFFERENCE, PRODUCT, SELECT, PROJECT, JOIN and DIVIDE.

UNION: This operation combines all the rows from two tables, excluding the rows which are having duplicate records. Here both the table must have the same fields and also share same number of columns. The example of union operation is shown in the figure-12:

Pro_id	P_name	Price
1	P1	250
2	P2	300
3	P3	350

Figure 12 (a)

UNION

Pro_id	P_name	Price
4	P4	400
5	P5	450
1	P1	250

Figure 12 (b)

Pro_id	P_name	Price
1	P1	250
2	P2	300
3	P3	350
4	P4	400
5	P5	450

Figure 12 UNION operation

INTERSECT: This operation displays only the records that are common on both the tables. The result of the intersection operation is given below:

FNAME
Tarun
Ravi

INTERSECT

FNAME
Tarun
Sam

OUTPUT

FNAME
Tarun

Figure-13Intersect operation

DIFFERENCE: It displays all the records in one table that are not found in another table. The result of the difference operation is shown below:

Product
Pen
Pencil
Ruler

DIFFERENCE

Product
Pen
Pencil

OUTPUT

Product
Ruler

Figure-14DIFFERENCE operation

PRODUCT: The product operation results in all the possible pair of rows from the two tables. This operation is also known as Cartesian product operation. For example if one table has 3 records and an other table has 2 records the product operation will yield 6 records. The output of product operation is shown below, where product operation is performed between Product table and Supplier table:

Pro_id	P_name	Price
P1	A	250
P2	B	300
P3	C	350

PRODUCT

SUP_id	S_Name	CITY
S1	RKT	AHM
S2	MBD	AHM

Pro_id	P_name	Price	SUP_id	S_Name	CITY
P1	A	250	S1	RKT	AHM
P1	A	250	S2	MBD	AHM
P2	B	300	S1	RKT	AHM
P2	B	300	S2	MBD	AHM
P3	C	350	S1	RKT	AHM
P3	C	350	S2	MBD	AHM

Figure -15The result of PRODUCT operation

SELECT: This operations displays all the records from the given table that satisfies a given criteria. This operation is also known as RESTRICT operation. For example suppose we want to list all the records from the above table where the price of the product is greater than 350, then the output of select operation is shown in the figure 17.

Pro_id	P_name	Price
P1	A	250
P2	B	300
P3	C	350
P4	D	400
P5	E	450

SELECT ALL(Price>350)

Pro_id	P_name	Price
P4	D	400
P5	E	450

Figure -16SELECT operation

PROJECT: This operation yields all the values for the selected attributes, which is a vertical subset of a given table. The result of PROJECT operation is shown in the figure 17:

Pro_id	P_name	Price
P1	A	250
P2	B	300
P3	C	350
P4	D	400
P5	E	450

PROJECT Price Yields

Price
250
300
350
400
450

Figure -17PROJECT operation

JOIN: Join allows information to be combined from two or more tables. There are several forms of join that are explained below.

A natural join links the tables by selecting only those rows with the common values in their common attribute, which is a three step process. First a PRODUCT operation is implemented among the tables included in the join. Secondly a SELECT operation is performed on the output to get the rows for which foreign key is present. And finally PROJECT operation is performed on the results of second operation to get the selected attributes and eliminate the duplicate tuples. The ultimate outcome of the natural join produces a set of a record that does not include matchless pairs and offer only the copies of the matches. Example of natural join and its operations are explained in the figures given below:

CUSTOMER TABLE			
CUST_ID	**NAME**	**PINCODE**	**A_ID**
C001	Sanjay	382330	A001
C002	Rahul	382421	A002
C003	Pankti	358965	A003
C004	Prachi	365898	A001

AGENT TABLE	
A_ID	**A_NAME**
A001	Hari
A002	Jay
A003	Om

Table-7 Sample tables considered for join illustrations

CUST_ID	NAME	PINCODE	CUSTOMER.A_ID	AGENT. A_ID	A_NAME
C001	Sanjay	382330	A001	A001	Hari
C001	Sanjay	382330	A001	A002	Jay
C001	Sanjay	382330	A001	A003	Om
C002	Rahul	382421	A002	A001	Hari
C002	Rahul	382421	A002	A002	Jay
C002	Rahul	382421	A002	A003	Om
C003	Pankti	358965	A003	A001	Hari
C003	Pankti	358965	A003	A002	Jay
C003	Pankti	358965	A003	A003	Om
C004	Prachi	365898	A001	A001	Hari
C004	Prachi	365898	A001	A002	Jay
C004	Prachi	365898	A001	A003	Om

Table-8Natural Join, Step 1: PRODUCT

The next operation performed in the natural join is a SELECT operation that is shown in the Table-9

CUST_ID	NAME	PINCODE	CUSTOMER.A_ID	AGENT. A_ID	A_NAME
C001	Sanjay	382330	A001	A001	Hari
C002	Rahul	382421	A002	A002	Jay
C003	Pankti	358965	A003	A003	Om
C004	Prachi	365898	A001	A001	Hari

Table-9 Natural Join, Step 2: SELECT

Finally the last operation implemented in natural join is PROJECT that is shown in the Table-10

CUST_ID	NAME	PINCODE	AGENT. A_ID	A_NAME
C001	Sanjay	382330	A001	Hari
C002	Rahul	382421	A002	Jay
C003	Pankti	358965	A003	Om
C004	Prachi	365898	A001	Hari

Table-10 Natural Join, Step 2: PROJECT

Another form of join is known as equijoin that links the tables on the basics of equality condition that compares specific attributes of each table. Here the output does not eliminate the duplicate column values. The equijoin takes the name from the operator that it uses, if any other comparison operator is used, the join is called a theta join.

Lastly the outer join, in which the matched pairs would be retained and any unmatched values in the other table would be left null.

DIVIDE: This operation uses one single-column table as the divisor and one two attribute table as the dividend. The tables used in this operation must have an attribute in common.

Key	Location
A	34
B	45
C	25
C	36
D	25
D	72
C	12

DIVIDE

Key
C
D

Location
25

Figure 18 Location Table is the outcome of the DIVIDE operations

Here the first table is divided by second table, where both the tables share a common attribute "KEY" and does not share LOCATION. The output yields only the value that is associated with both "C" and "D".

4.7 ATTRIBUTES

Attributes are considered to be the characteristics of the entities. For example the CUSTOMER entity consists of many attributes like CUST_ID, NAME, PINCODE, EMAIL etc. Here in this section we will discuss about various points to be kept in mind while deciding the attributes in a given entity.

Required and Optional attributes: A required attribute is an filed that must have a value or which cannot be left null. For example CUST_ID and NAME are required attributes in the CUSTOMER table. On the contrary a customer may have an email or may not so the field EMAIL in the CUSTOMER table is an optional attribute as it can be left null.

Domains: All the attributes have their domain, which means a set of possible values that can be accepted by that particular filed. For example minimum and maximum value for semesters in the MSc(IT) course can be between one and four. So the domain of possible values for the field semester is either 1/2/3/4.

Primary key: Primary key is the identifier that is used to identify each record or tuple uniquely. Also it cannot be null. For example CUST_ID in the CUSTOMER table is a primary key that uniquely identifies each customer's record and which cannot be null.

Composite keys: When we use more than one identifier or primary key to uniquely identify a record in a table, it is known as a composite key. For example CUST_ID and ACCOUNT_NUM can be combined to create a composite key as a customer may have different types of account in a bank,

Composite and simple attributes: A composite attribute is not be baffled with composite key. It is an attribute that can be further sub divided to yield additional attributes. For example an attribute FULL_NAME can be further sub divided into FIRST_NAME, MID_NAME and LAST_NAME. A simple attribute cannot be further sub divided. For example gender, age etc.

Single-valued attributes: An attribute that can have only single value is known as single valued attribute. For example AADHAR number of any Indian citizen is considered to be a single-valued attribute.

Multivalued attributes: Those attributes that can have multiple values for example color of a car, degree of a student, area of interest of a candidate, hobbies etc are considered to be the multivalued attributes.

Derived attributes: An attributes value that can be calculated from other attributes value is known as derived attribute. For example the attribute AGE can be derived from the date of birth field. Similarly amount of GST to be paid, percentage of a student etc are the examples of derived attributes.

4.8 RELATIONSHIPS CONTAINED IN RELATIONAL DATABASE

Relationships that are defined in relational database are of three types:

- One-to-many (1:M)
- One-to-one (1:1)
- Many-to-many (M:N)

The 1:M relationship: The 1:M relationship is the relational database standard. To this how this relationship is modeled and implement let us consider a simple example of COUNTRY and STATE entity.

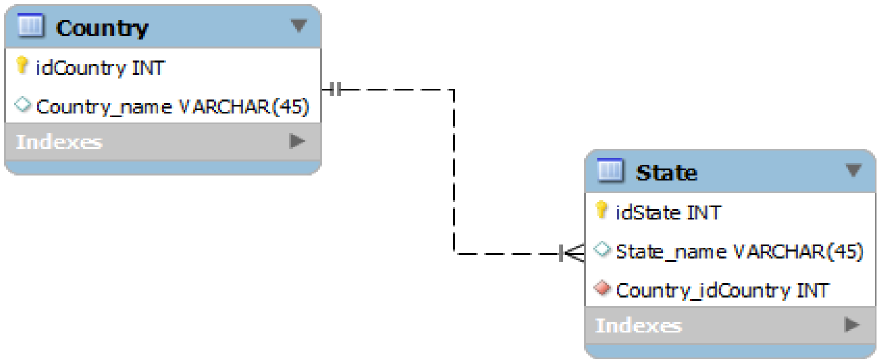


Figure-19:M relationship between Country and State table

As shown the figure 19 the one COUNTRY can have many STATES, so there is a one-to-many relationship between two tables.

The 1:1 relationship: This relationship represents that one entity can be related to only one another entity and vice versa. For example one department chair-a professor-can chair only one department and one department can have only one department chair.

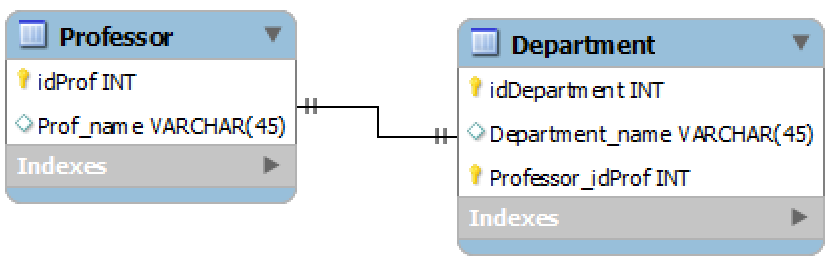


Figure-20 1:1 relationships between Professor and Department

The M:N relationship: A M:N relationship is not directly supported in the relational database environment. As ample example of M:N relationship can be considered between MOBILE and FEATURES tables. Here one MOBILE can have many features, also the same feature can be there in many MOBILES.

The way to implement M:N relationship in relational database environment is to change the M:N relationship to *two* 1:M relationship. This can be done by adding a third associative entity or a bridge table between two tables. Figure 1. 31 represents the solution to the given problem. Here the bridge table is "Mobile_has_feature", which specifies which mobile has which features.

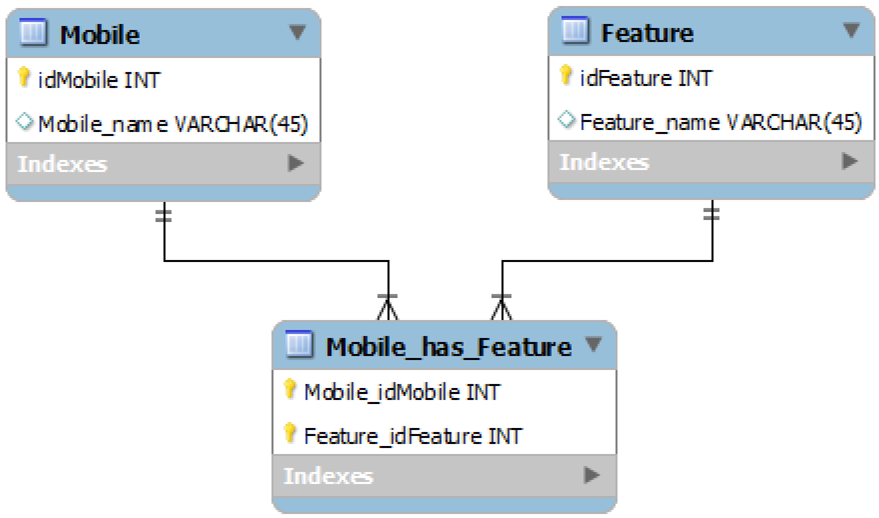


Figure-21 Changing the M:N relationship to two 1:M relationship

4.9 CONNECTIVITY AND CARDINALITY

Cardinality signifies the minimum and the maximum number of entity occurrences associated with one occurrence of the related entity. In entity relationship modeling it is represented by using the format (n,m), where the first parameter represents minimum number of linked entities and the second parameter represents the maximum number of entity occurrences. The below figure shows the example of PROFESSOR and CLASS entity.

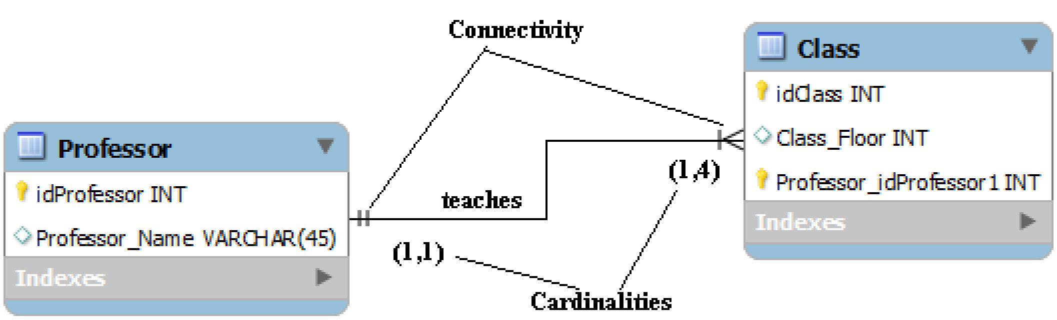


Figure-22 Connectivity and Cardinality

4.10 RELATIONSHIP STRENGTH

The notion of relationship strength is based on how the primary key of a related entity is defined. There are basically two types of relationship strength weak and strong relationships which are discussed below:

Weak Relationships: It is also known as Non-identifying relationship. It exists when primary key of a related entity does not contain a primary key component of the parent entity. By default relationships are recognized by having the primary key of the parent entity appear as a foreign key on the related entity. For example, suppose that the COURSE and CLASS entities are defined as:

COURSE (**CRS_CODE**, DEPT_CODE, CRS_DESC, CRS_CREDIT)

CLASS (**CLASS_CODE,** CRS_CODE, CLASS_SECTION, PROF_ID,CLASS_TIME)

In this case a weak relationship exists between the above two entities because the CRS_CODE in CLASS entity is only an foreign key.

Strong Relationships: A strong relationship is also known as identifying relationship. It exists when the primary key of a related entity contains primary key component of a parent entity. For example if we consider the COURSE and CLASS entities as:

COURSE (**CRS_CODE**, DEPT_CODE, CRS_DESC, CRS_CREDIT)

CLASS (**CRS_CODE, CLASS_SECTION**, PROF_ID, CLASS_TIME)

This indicates a strong relationship exists between the entities COURSE and CLASS, because the CLASS entity contains a composite primary key of CRS_CODE and CLASS_SECTION.

4.11 RELATIONSHIP DEGREE

A relationship degree specifies the number of entities that are associated with a relationship. They are of several types like unary, binary, ternary and higher degree relationship that are discussed below:

Unary relationships: An example of the unary relationship is shown in the figure 1.33, where an Employee entity is a supervisor for one or more workers who are again employees within that entity. Such a relationship is also known as recursive relationships. Recursive relationships exits between the occurrences of the same entity set.

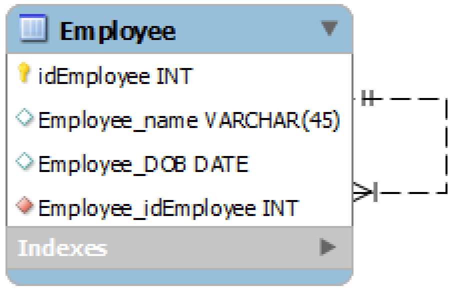


Figure-23 Unary relationship

Binary relationships: A binary relationship exists when there are two entities that are related with each other as shown in the figure 1.34. It is the most frequent relationship that exists in the relational database. A basic example of two Entities CITY and AREA table is shown below that are having one-to-many relationship.

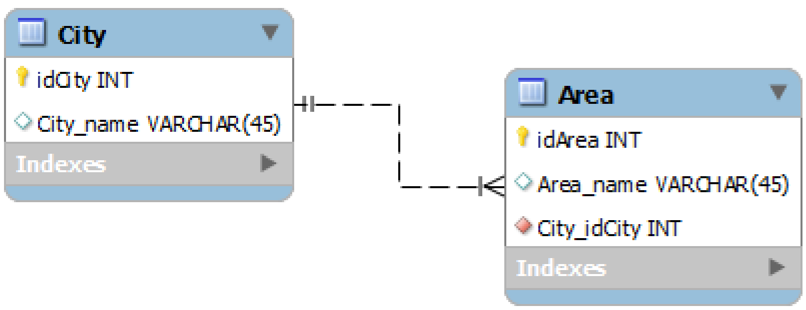


Figure-24 Binary relationship

Ternary and Higher degree relationships: A ternary relationship involves relationship among three different entities. Let's take an example of three entities DOCTOR, PATIENT and MEDICINE. Here the doctor gives one or more prescriptions to the patients. Patients can visit one or more doctors and get different prescriptions. One medicine can be there in one or more prescriptions that are given by doctor to patients. An example of ternary relationship is as shown in figure 1.35

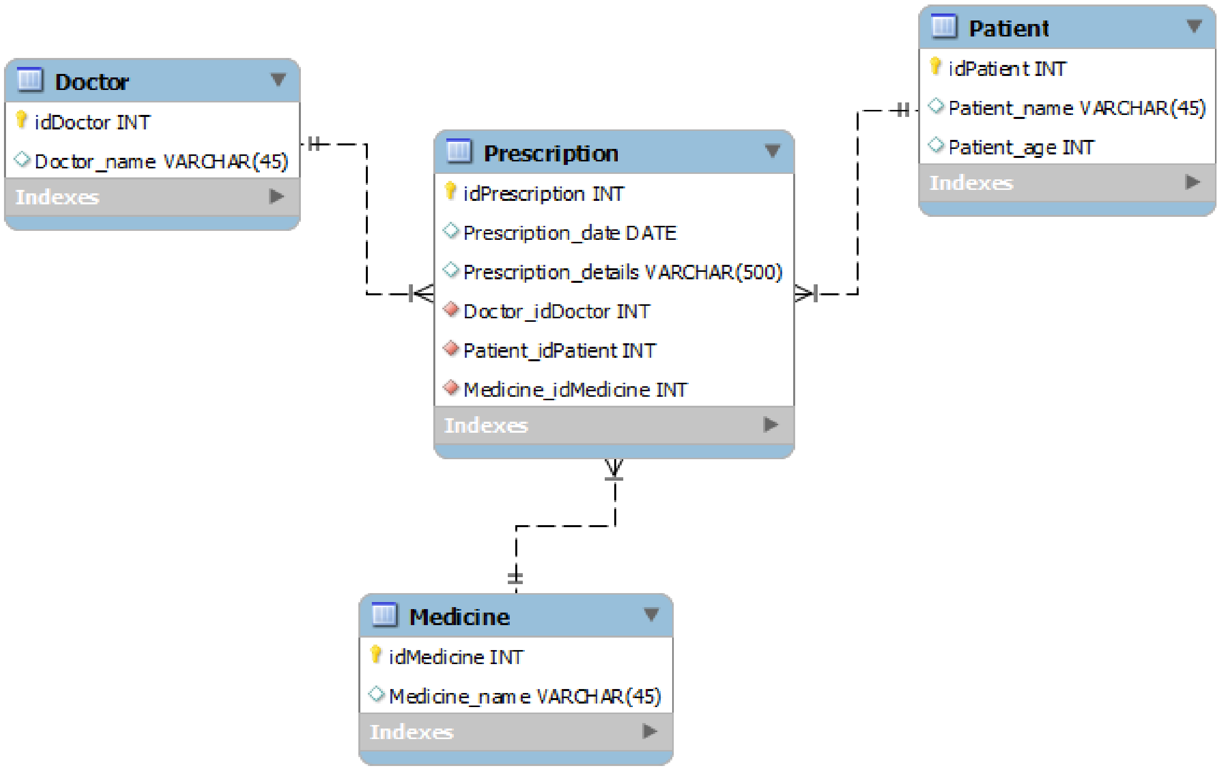


Figure-25 Ternary relationship

4.12 DATABASE DESIGN PROCESS

Database design is a procedure of creating a complete data model of a database consisting of all the logical and physical design alternatives and physical storage considerations needed to create a design of a database. It should always reflect the information system and should undergo evaluation and revision within a framework known as Database life cycle. There are two methods of database design:

✓ **Top-down vs. Bottom-up design**

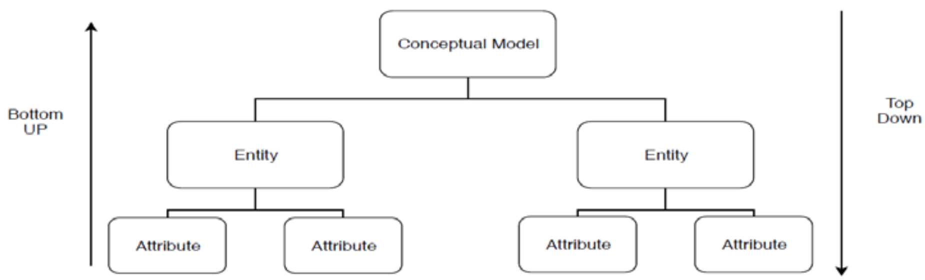


Figure-26 Top-down vs. Bottom-up design

In top dow n appr oach we identify t he dat aset and define t he dat a el ements. In bottom-up approach we identify the data elements first and then we group them into datasets.

- ✓ **Centralized vs. Decentralized design**

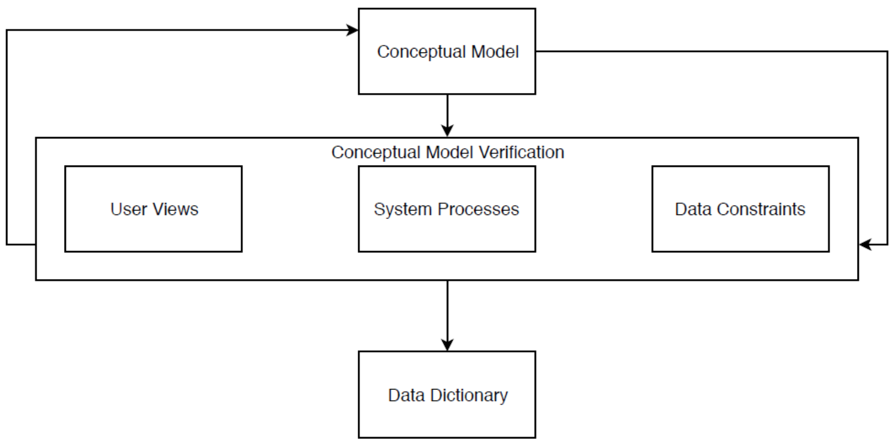


Figure-27 A centralized design approach

In centralized database design is conducted by a single person or a small team as s hown i n t he f igure 1. 37 o n t he c ontrary i n dec entralized database design l arge number of relationship and complex relations exits and are spread across multiple sites as shown in the figure 1.38

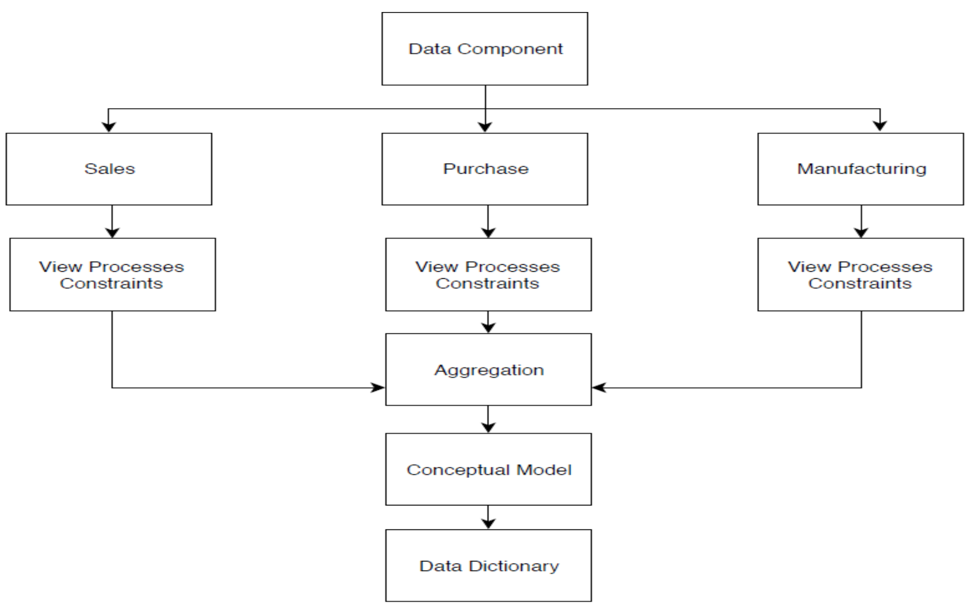


Figure-28 A decentralized design approach

DATABASE LIFE CYCLE (DBLC):

Phase 1: Database Initial Study: In the initial study we analyze the organization structure and its operating environment. We define the problem and list all the constraints. We need to also state the main objectives of the proposed system along with its scope and boundaries.

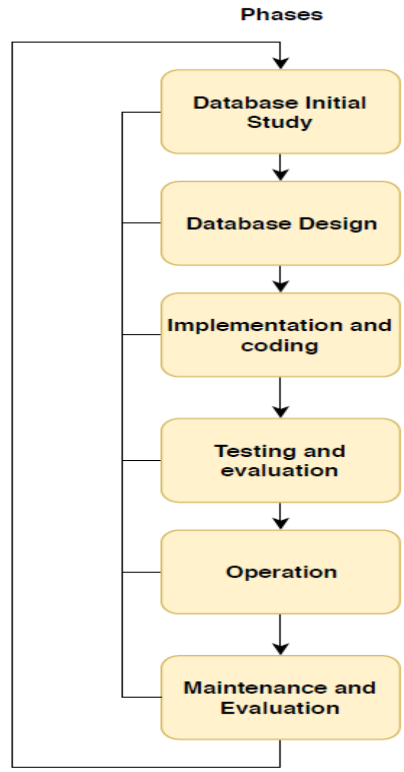


Figure-29 Phases in DBLC

Phase 2: Database Design: It is the most critical phase where the DBA has to focus on data requirements create a conceptual design, Select the DBMS software, create a logical design and create a physical design.

Conceptual Design: In conceptual design we map the database with the real world entities. Here we perform data analysis and requirements, develop and ER and normalize to its required forms and lastly we verify the data model that is developed.

DBMS Software selection: The factors that must be considered at the time of DBMS software selection are:

- ✓ Underlying model of database
- ✓ DBMS features and tools

- ✓ DBMS hardware requirements
- ✓ Portability of the DBMS
- ✓ COST

Logical Design: The logical design translates the conceptual design into internal model. Here the logical model design components are Tables, Indexes, Views, Transactions etc.

Physical Design: In physical design we need to specify the data storage and access characteristics because this becomes very difficult in case of distributed systems.

Phase 3: Implementation and coding: This phase includes creation of special storage constructs for the end user tables. It also gives solution to other issues like performance, security, backup and recovery, maintaining industry standards and managing concurrency controls.

Phase 4: Testing and evaluation: In this phase the database is tested and fine tuned for performance, integrity, concurrent access and security constraints. This phase is implemented in parallel with application programming. If the testing fails then following actions are taken:

- ✓ Fine tuning based on reference models
- ✓ Alterations in the logical design
- ✓ Updating in the physical design
- ✓ Modernize or change the DBMS software or hardware in which its implemented

Phase 5: Operation: In this phase database is considered to be operational and the process of system evaluation begins. During this phase some unforeseen problems may occur and demand for a change.

Phase 6: Maintenance and Evaluation: In this phase we implement different maintenance techniques like preventive maintenance, corrective maintenance, adaptive maintenance, assignment of access permission, producing database statistics for monitoring performance, conducting security audits based on system-generated statistics.

4.13 ANOMALIES IN DATABASE

Anomalies are in fact troubles that can arise in poorly designed, non-normalized databases. Non-normalized databases are those databases which don't follow database standard rules in order to design and develop it. There are several categories of anomalies that can exist while referencing attributes in the related tables. Suppose we consider here two entities as STUDENT and COURSE and the sample records are shown below:

STUDENT_ID	NAME	EMAIL	AGE
S001	Vivek	v@gmail.com	25
S002	Abhi	ab@ymail.com	27
S003	Aniket	an@yahoo.com	32

Figure 1.40 Student table

COURSE_ID	NAME	STUDENT_ID
C1	Python Programming	S001
C2	Networking	S003
C3	Java Programming	S001

Figure 1.41 Course table

Insertion anomaly: If a record is inserted in a referenced attribute and the corresponding foreign key is not present in the primary table (STUDENT), it will result in insertion anomaly. For example if we try to insert S005 in the COURSE table, it will not permit.

Deletion and updation anomaly: If a record is deleted or edited from referenced relation and referenced field value is used by referencing attribute in associated relation, it will not permit deleting the record from referenced association. For example if we try to delete the record from the STUDENT table where STUDENT_ID is S003, it will not permit to delete the record. In order to avoid such a situation we can use CASCADE UPDATE and CASCADE DELETE while query processing.

4.14Check your progress

1. Define table and explain its characteristics by giving examples.
2. List and explain the importance of integrity policies in relational DBMS.
3. Discuss relational set operators in detail.
4. What are the points to be kept in mind while deciding the attributes for a given entity?
5. Write a short note on relationship degree.
6. Discuss the database design process.
7. List and explain the anomalies faced in the database.

Block-2

Relational Data Model

and

Introduction to Oracle

Server

Unit 1: Functional Dependency and Normalization

1

Unit Structure

1.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this chapter is to make the students,

- To learn and understand Dependencies and how to define it.
- To understand the Armstrong's Axioms of FDs.
- To understand the decomposition process of database relation.
- To learn normalization process and different normal forms.

Outcome:

At the end of this unit,

- Students will be completely aware with process of Dependencies and its different types like Function Dependencies, Fully Functional Dependencies, Multivalued Dependencies, Join Dependencies etc.
- Students will come to know the decomposition process and its types.
- Students will come to know normalization and different normal forms.

1.2 INTRODUCTION

Functional dependencies (FDs) play a key role in differentiating good database designs from database design. A functional dependency is a type of constraint that is a generalization of the notion a key Functional dependencies. FD's are constraints on well-formed relations and represent formalism on the infrastructure of relation. The determination of functional dependencies is an important part of designing databases in the relational model, and in database normalization and denormalization. The functional dependencies, along with the attribute domains, are selected so as to generate constraints that would exclude as much data inappropriate to the user domain from the system as possible.

Normalization (NF) is a systematic way of ensuring that a database structure is suitable for general-purpose querying and free of certain undesirable characteristics like insertion, update, and deletion anomalies; that could lead to a loss of data integrity. The normal forms of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The normal forms are applicable to individual tables; to say that an entire database is in normal form n is to say that all of its tables are in normal form n.

1.3 FUNCTIONAL DEPENDENCY

A functional dependency (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X, if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below:

X → Y Or X à Y

The left side of the above FD diagram is called the Determinant (X), and the right side is the Dependent (Y).

X	Y
1	1
2	4
3	9
4	16
2	4
7	9
Table: A	

X	Y
1	1
2	4
3	9
4	16
2	10
7	9
Table: B	

Above Table: A illustrates that X → Y, since for each value of X there is associated one and only one value of Y. While Table: B illustrates that X does not functionally determine Y, since for X = 2 there is associated more than one value of Y (4, 10).

Example: Consider the database having following tables.

SNo	SName	Status	City
S1	Nilesh	20	Ahmedabad
S2	Vinod	10	Patan
S3	Rahul	20	Ahmedabad
S4	Jayesh	20	Surat

Table: Supplier

Here, if we know the value of SNo, We can obtain value of SName, Status and City. So, we can say that SName, Status and City are functionally depends on SNo. FD is represented as: SNo →**{ SName,Status,City}**

SNo	PNo	Qty
S1	P1	270
S1	P2	300
S1	P3	700
S2	P1	270
S2	P2	450
S3	P2	280

Table: Shipment

In this case Qty is FD on combination of SNo and PNo, because each combination of SNo and PNo results only one Qty. FD is represented as: **{SNo, PNo} → Qty**

1.3.1. FULLY FUNCTIONAL DEPENDENCY (FFD)

Fully Functional Dependence (FFD) is defined, as Attribute Y is FFD on attribute X, if it is FD on X and not FD on any proper subset of X. For example, in relation Supplier, different cities may have the same status. It may be possible that cities like Ahmedabad, Surat may have the same status 20. So, the City is not FD on Status.

But, the combination of SNo, Status can give only one corresponding City, because SNo is unique. Thus,

{SNo, Status} → City

It means city is FD on composite attribute (SNo, Status) however City is not fully functional dependent on this composite attribute, which is explained below:

{SNo , Status} → City

X Y

Here Y is FD on X, but X has two proper subsets SNo and Status; city is FD on one proper subset of X. **SNo → City**

According to FFD definition Y must not be FD .on any proper subset of X, but here City is FD in one subset of X i.e. SNo, so City is not FFD on (SNo, Status)

1.3.2. ARMSTRONG'S AXIOMS OF FUNCTIONAL DEPENDENCIES (INFERENCE RULES)

A set of rules that may be used to infer additional dependencies was proposed by William W. Armstrong in 1974. These rules (or axioms) are a complete set of rules in· that all possible functional dependencies may be derived from them. Below given are the three most important rules for FD:

- **Reflexive Rule:** If X is a set of attributes and Y is subset of X, then X holds a value of Y.
- **Augmentation Rule:** When x → y holds, and c is attribute set, then ac → bc also holds. That is adding attributes which do not change the basic dependencies.
- **Transitivity Rule:** This rule is very much similar to the transitive rule in algebra. if x→ y holds and y → z holds, then x → z also holds.

Further axioms may be derived from the above although the above three axioms are sound and complete in that they do not generate any incorrect functional dependencies (soundness) and they do generate all possible functional dependencies that can be inferred from F (completeness). The most important additional axioms are:

a. **Union Rule:** If X → Y and X → Z hold, then X → YZ holds.

b. **Decomposition Rule:** If X → YZ holds, then so do X → Y and X → Z.

A. Trivial Functional Dependency

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute. So, X → Y is a trivial functional dependency if Y is a subset of X.

Example: Consider a Employee table

EmpId	EmpName	EmpContact
1001	Jayesh Patel	8625610860
1002	Viral Vyas	7300456780
1003	Chirag Prajapati	6625674610

Table: Employee

{EmpId,EmpName} → EmpName is a trivial functional dependency as a EmpName is a subset of {EmpId,EmpName}. If we knows the value of EmpId and EmpName then the value of EmpId can be uniquely determined. Also, EmpId →EmpId & EmpName → EmpName are trivial dependencies too.

B. Non-Trivial Functional Dependency

If a functional dependency X → Y holds true where Y is not a subset of X then this dependency is called Non-Trivial functional dependency.

Example: Consider a Employee table. Following functional dependencies are Non-trivial.

EmpId → EmpName (EmpName is not a subset of EmpId)
EmpId → EmpContact (EmpContact is not a subset of EmpId)

If a functional dependency X → Y holds true where X intersection Y is null then this dependency is called **completely Non-Trivial FD**.

C. Transitive Functional Dependency

Transitive Functional Dependency happens when it is indirectly formed by two functional dependencies. This dependency can only occur in a relation with minimum three attributes.

Example: Consider a Employee table

EmpId → EmpName (If we know EmpId, we know its Name)
EmpName → EmpContact (If we know EmpName, we know its Contact)

Therefor as per rule of transitive dependency; **EmpId → EmpContact** should hold, that make sense if we know the EmpId, we can know his Contact.

1.4 DECOMPOSITION

A functional **decomposition** is the process of breaking down the functions of an organization into progressively greater levels of detail. The decomposition of a relation scheme R consists of replacing the relation schema by two or more relation schemas that each contain a subset of the attributes of R and together include all attributes in R. Decomposition helps in eliminating some of the problems of bad design such as redundancy, inconsistencies and anomalies.

Lossy Decomposition: **The decomposition of relation R into R1 and R2 is** lossy **when the join of R1 and R2 does not yield the same relation as in R. One of the disadvantages of decomposition into two or more relational schemes (or tables) is that some information is lost during retrieval of original relation or table. Spurious rows are generated when a natural join is applied to the relations in the decomposition.**

Lossless Join Decomposition: **The decomposition of relation R into R1 and R2 is lossless when the join of R1 and R2 yield the same relation as in R. A relational table is decomposed into two or more smaller tables, in such a way that the designer can capture the precise content of the original table by joining the decomposed parts. This is called lossless-join (or non-additive join) decomposition. Spurious tuples are not generated when a natural joined is applied to the relations in the decomposition.**

Dependency-Preserving Decomposition: The dependency preservation decomposition is another property of decomposed relational database schema D in which each functional dependency X -> Y specified in F either appeared directly in one of the relation schemas Ri in the decomposed D or could be inferred from the dependencies that appear in some Ri.

Decomposition D = { R_1 , R_2, R_3,,.., ,R_m} of R is said to be dependency-preserving with respect to F if the union of the projections of F on each R_i , in D is equivalent to F. The dependencies are preserved because each dependency in F represents a constraint on the database. If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

1.5 CLOSURE SET OF FUNCTIONAL DEPENDENCIES

A **Closure** is a set of FDs is a set of all possible FDs that can be derived from a given set of FDs. It is also referred as a **complete** set of FDs. If F is used to donate the set of FDs for relation R, then a closure of a set of FDs implied by F is denoted by F^+.

Let's consider the set F of functional dependencies given below:

F = {A -> B, B -> C, C -> D}

from F, it is possible to derive following dependencies.

A -> A ...By using Rule-4, Self-Determination.

A -> B ...Already given in F.

A -> C ...By using rule-3, Transitivity.

A -> D ...By using rule-3, Transitivity.

Now, by applying Union Rule, it is possible to derive A^+ -> ABCD and it can be denoted using A -> ABCD. All such type of FDs derived from each FD of F form a closure of F.

Steps to determine F^+:

- Determine each set of attributes **X** that appears as a left hand side of some FD in F.
- Determine the set $\mathbf{X^+}$ of all attributes that are dependent on **X**.
- $\mathbf{X^+}$ represents a set of attributes that are functionally determined by X based on F. And, X^+ is called the **Closure of X under F.**
- All such sets of $\mathbf{X^+}$, in combine, Form a closure of F.

Find Candidate Keys

A super key is a set of attributes whose closure is the set of all attributes. In other words, a super key is a set of attributes you can start from, and following functional dependencies, will lead you to a set containing each and every attribute. A candidate key is a minimal super key. The first step to finding a candidate keys, is to find all the super keys.

Example: Given the Relation R with attributes ABCDE. You are given the following dependencies: A -> B, BC -> E, and ED -> A.

Since we have the functional dependencies: A -> B, BC -> E, and ED -> A, we have the following super keys:

- ABCDE (All attributes is always a super key)
- BCED (We can get attribute A through ED -> A)
- ACDE (Just add B through A -> B)
- ABCD (Just add E through BC -> E)
- ACD (We can get B through A -> B, and then we can get E through BC -> E)
- BCD (We can get E through BC -> E, and then A from ED -> A)
- CDE (We can get A through ED -> A and then B from A -> B)

We can see that only the last three are candidate keys. Since the first four can all be trimmed down. But we cannot take any attributes away from the last three super keys and still have them remain a super key. Thus the candidate keys are: **ACD, BCD, and CDE**.

1.6 NORMALIZATIONS

Database Normalization is a technique that helps in designing the schema of the database in an optimal manner so as to ensure the above points. The core idea of database normalization is to divide the tables into smaller sub tables and store pointers to data rather than replicating it.

Normalization results in decomposition of the original relation. It should be noted that decomposition of relation has to be always based on principles, such as functional dependence, that ensure that the original relation may be reconstructed from the decomposed relations if and when necessary. Careless decomposition of a relation can result in loss of information.

1.6.1 THE FIRST NORMAL FORM (1NF)

Definition: A relation (table) is in 1NF if

1. There are no duplicate rows or tuples in the relation.
2. Each data value stored in the relation is single-valued
3. Entries in a column (attribute) are of the same kind (type).

In a 1 NF relation the order of the tuples and attributes does not matter. The first requirement above means that the relation **must have a key**. The key may be single attribute or composite key. The first normal form defines only the basic structure of the relation and does not resolve the anomalies.

The relation STUDENT is in 1NF. The primary key of the relation is (Sno+Cno).

STUDENT

Sno	Sname	Address	Cno	Cname	Instructor	Office
101	Viral	Ahmedabad	MCIT–101	OOPS with Java	Amit Kumar	102
101	Viral	Ahmedabad	MCIT–102	RDBMS	Bhavesh Patel	105
101	Viral	Ahmedabad	MCIT-104	Networking	Jignesh Patel	103
102	Dashrath	Ahmedabad	MCIT-104	Networking	Jignesh Patel	103

1.6.2 THE SECOND NORMAL FORM (2NF)

Definition: A relation is in 2NF if it is in 1 NF and every non-key attribute is fully dependent on each candidate key of the relation.

Some of the points that should be noted here are:

- A relation having a single attribute key has to be in 2NF.
- In case of composite key, partial dependency on key that is part of the key is not allowed.
- 2NF tries to ensure that information in one relation is about one thing
- Non-key attributes are those that are not part of any candidate key.

These FDs of relation STUDENT can also be written as:

Sno	→	Sname, Address	(1)
Cno	→	Cname, Instructor	(2)
Instructor	→	Office	(3)

For the 2NF decomposition, we are concerned with the FDs (1) and (2) as above as they relate to partial dependence on the key that is (Sno + Cno). To convert the relation into 2 NF, let us use FDs. As per FD (1) the Student number uniquely determines student name and address, so one relation should be:

STUDENT1 (Sno, Sname, Address)

Sno	Sname	Address
101	Viral	Ahmedabad
102	Dashrat	Ahmedab

	h	ad

We f ind i n F D (2) t hat C ourse c ode at tribute uniquely det ermines t he nam e of instructor (refer t o F D 2(a)). Also t he F D (3) m eans t hat name of t he i nstructor uniquely determines office number. This can be written as:

Cno → Instructor (2 (a)) (without Cname)
Instructor → Office (3)
Cno → Office (This is transitive dependency)

Thus, FD (2) now can be rewritten as:

Cno → Cname, Instructor, Office (2')

This FD, now gives us the second decomposed relation:

COU_INST (Cno, Cname, Instruction, Office)

Cno	Cname	Instructor	Office
MCIT–101	OOPS with Java	Amit Kumar	102
MCIT–102	RDBMS	Bhavesh Patel	105
MCIT-104	Networking	Jignesh Patel	103

We hav e s uper F Ds as , bec ause (Sno + C no) i s t he pr imary k ey of t he r elation STUDENT:

Sno, Cno → ALL ATTRIBUTES

All the attributes except for the key attributes that are Sno and Cno, however, are covered on the right side of the FDs (1) (2) and (3), thus, making the FD as redundant. But in any case we have to have a relation that joins the two decomposed relations. This relation would cover any attributes of Super FD that have not been covered by the decomposition and the key attributes. Thus, we need to create a joining relation as:

COURSE_STUDENT (Sno, Cno)

Sno	Cno
101	MCIT–101
101	MCIT–102
101	MCIT-104
102	MCIT-104

So, the relation STUDENT in 2NF form would be, STUDENT1, COU_INST AND COURSE_STUDENT.

1.6.3 THE THIRD NORMAL FORM (3NF)

Definition: A relation is in third normal form, if it is in 2NF and every non-key attribute of the relation is non-transitively dependent on each candidate key of the relation**.**

Let us reconsider the relation 2NF (b)
COU_INST (Cno, Cname, Instruction, Office)

Assume that Cname is not unique and therefore Cno is the only candidate key. The following functional dependencies exists

Cno	→	Instructor (2 (a)) (without Cname)
Instructor	→	Office (3)
Cno	→	Office (This is transitive dependency)

The relation is however not in 3NF since the attribute 'Office' is not directly dependent on a ttribute ' Cno' but i s t ransitively dependent on it an d s hould, t herefore, be decomposed as it has all the anomalies. We need to decompose the relation 2NF(b) into the following two relations:

COURSE:

Cno	**Cname**	**Instructor**
MCIT–101	OOPS with Java	Amit Kumar
MCIT–102	RDBMS	Bhavesh Patel
MCIT-104	Networking	Jignesh Patel

INST:

Instructor	**Office**
Amit Kumar	102
Bhavesh Patel	105
Jignesh Patel	103

Two relations and 2NF (a) and 2NF (c) are already in 3NF. Thus, the relation STUDENT in 3NF would be:

STUDENT1 (Sno, Sname, Address)
COURSE (Cno, Cname, Instructor)
INST (Instructor, Office)
COURSE_STUDENT (Sno, Cno)

The 3NF is usually quite adequate for most relational database designs. There are however some situations where a relation may be in 3NF, but have the anomalies. For example, consider the relation NEWSTUDENT (Sno, Sname, Cno, Cname) having the set of FDs:

Sno → Sname
Sname → Sno
Cno → Cname
Cname → Cno

The relation is in 3NF. All the attributes of this relation are part of candidate keys, but have dependency between the non-overlapping portions of overlapping candidate keys. Thus, the 3NF may not eliminate all the redundancies and inconsistencies. Thus, there is a need of further Normalization using the BCNF.

1.6.4 BOYCE-CODD NORMAL FORM (BCNF)

The relation NEWSTUDENT (Sno, Sname, Cno, Cname) has all attributes participating in candidate keys since all the attributes are assumed to be unique. Since the relation has no non-key attributes, the relation is in 2NF and also in 3NF.

Definition: A relation is in BCNF, if it is in 3NF and if every determinant is a candidate key.

- A determinant is the left side of an FD
- Most relations that are in 3NF are also in BCNF. A 3NF relation is not in BCNF if all the following conditions apply.
  1. The candidate keys in the relation are composite keys.
  2. There is more than one overlapping candidate keys in the relation, and some attributes in the keys are overlapping and some are not overlapping.
  3. There is a FD from the non-overlapping attribute(s) of one candidate key to non-overlapping attribute(s) of other candidate key.

NEWSTUDENT (Sno, Sname, Cno, Cname) Set of FDs:

Sno → Sname (1)
Sname → Sno (2)
Cno → Cname (3)
Cname → Cno (4)

The relation although in 3NF, but is not in BCNF and can be decomposed on any one of the FDs in (1) & (2); and any one of the FDs in (3) & (4) as:

STUD1 (Sno, S name)
COUR1 (Cno, C name)

The third relation that will join the two relation will be: ST_CO(Sno, Cno)

1.6.5 MULTIVALUED DEPENDENCIES AND 4TH NORMAL FORM

A. Multivalued Dependencies:

If two or more independent relation are kept in a single relation or we can say multivalue dependency occurs when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. Put another way, two attributes (or columns) in a table are independent of one another, but both depend on a third attribute. A multivalued dependency always requires at least three attributes because it consists of at least two attributes that are dependent on a third. A functional dependency is a special case of multivalued dependency. In a functional dependency X → Y, every x determines exactly one y, never more than one.

For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency. The table should have at least 3 attributes and B and C should be independent for A →→ B multivalued dependency. For example,

Person	Mobile	Food_Likes
Viral Vyas	989898009	Burger
Amit Patel	756427523	Pizza

Person →→ mobile, Person →→ food_likes

B. Fourth normal form (4NF):

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the BCNF. It states that, in addition to a database meeting the requirements of BCNF; it must not contain more than one multivalued dependency.

Properties:

A relation R is in 4NF if and only if the following conditions are satisfied:

1. It should be in the Boyce-Codd Normal Form (BCNF).
2. the table should not have any Multi-valued Dependency.

A table with a multivalued dependency violates the normalization standard of Fourth Normal Form because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, it is necessary to break this information into two tables.

Example:
Consider the database table:

Student (Sno,Sname):

Sno	Sname
101	Viral Vyas
102	Amit Patel

Course (Cno,Cname)

Cno	Cname
2001	MCA
2002	M.Sc.(IT)

When there cross product (Student X Course) is done it resulted in multivalued dependencies:

Sno	Sname	Cno	Cname
101	Viral Vyas	2001	MCA
101	Viral Vyas	2002	M.Sc.(IT)
102	Amit Patel	2001	MCA
102	Amit Patel	2002	M.Sc.(IT)

Multivalued dependencies (MVD) are:

SID →→ CID; SID →→ CNAME; SNAME →→ CNAME

1.6.6 JOIN DEPENDENCIES AND 5NF / PJNF

The fifth normal form deals with join-dependencies, which is a generalisation of the MVD. The aim of fifth normal form is to have relations that cannot be decomposed further. A relation in 5NF cannot be constructed from several smaller relations.

A relation R satisfies join dependency *(R1, R2, ..., Rn) if and only if R is equal to the join of R1, R2, ..., Rn where Ri are subsets of the set of attributes of R.

A relation R is in 5NF if for all join dependencies at least one of the following holds:

a) (R1, R2, ..., Rn) is a trivial join-dependency.
b) Every Ri is a candidate key for R.

An example of 5 NF can be provided by the relation employee that deals with emp_name, Projects and Programming languages.

emp_name	**projects**	**languages**
VIRAL	Proj_A	C
AMIT	Proj_A	Java
VIRAL	Proj_B	C
AMIT	Proj_B	C++

The relation above assumes that any employee can work on any project and knows any of the three languages. The relation also says that any employee can work on projects Proj_A, Proj_B, Proj_C and may be using a different programming languages in their projects. No employee takes all the projects and no project uses all the programming languages and therefore all three fields are needed to represent the information. Thus, all the three attributes are independent of each other.

The relation above does not have any FDs and MVDs since the attributes emp_name, project and languages are independent; they are related to each other only by the pairings that have significant information in them. For example, VIRAL is working on Project A using C languague. Thus, the key to the relation is (emp_name, projects, l anguages). T he r elation i s i n 4 NF, b ut s till s uffers f rom t he i nsertion, deletion, and update anomalies. However, the relation therefore cannot be decomposed in two relations.

(emp_name, project) and (emp_name, language)

The decomposition mentioned above will create tables as given below:

emp_project

emp_name	Projects
VIRAL	Proj_A
AMIT	Proj_A
VIRAL	Proj_B
AMIT	Proj_B

emp_language

emp_name	Languages
VIRAL	C
AMIT	Java
AMIT	C++

On taking join of these relations on emp_name it will produce the following result:

emp_name	projects	languages
VIRAL	Proj_A	C
AMIT	Proj_A	Java
AMIT	Proj_A	C++
VIRAL	Proj_B	C

AMIT	Proj_B	Java
AMIT	Proj_B	C++

Since the joined table does not match the actual table, we can say that it is a lossy decomposition. Thus, the expected join dependency expression; * ((emp_name, project), (emp_name, language)) does not satisfy the conditions of lossless decomposition. Hence, the decomposed tables are losing some important information.

1.6.7 PROJECT-JOIN NORMAL FORM

PJNFis defined using the concept of the join dependencies. A relation schema *R* having a set *F* of functional, multivalued, and join dependencies, is in PJNF (5 NF), if for all the join dependencies in the closure of F (referred to as *F*+) that are of the form *($R1, R2, \ldots, Rn$), where each $Ri \subseteq R$ and $R = R1 \cup R2 \cup \ldots \cup Rn$, at least one of the following holds:

- *($R1, R2, \ldots, Rn$) is a trivial join dependency.
- Every *Ri*is a superkey for *R*.

PJNF is also referred to as the Fifth Normal Form (5NF). Let us first define the concept of PJNF from the viewpoint of the decomposition and then refine it later to a standard form.

Definition 1: A JD *[R1, R2, . . . , Rn] over a relation R is trivial if it is satisfied by every relation r(R). The trivial JDs over R are JDs of the form *[R1, R2, . . . , Rn] where for some i the Ri = R.

Definition 2: A JD *[R1, R2, . . ., Rn] applies to a relation scheme R if R = R1 R2 …Rn.

Definition 3: Let R be a relation scheme having F as the set of FDs and JDs over R. R will be in project-join normal form (PJNF) if for every JD *[R1, R2, . . ., Rn] which can be derived by F that applies to R, the following holds:

- The JD is trivial, or
- Every Ri is a super key for R.

For a database scheme to be in project-join normal form, every relation *R* in this database scheme should be in project-join normal form with respect to *F*. The definition of *PJNF* as given above is a weaker than the original definition of *PJNF* given by Fagin. The original definition ensures enforceability of dependencies by satisfying keys, in addition to elimination of redundancy.

Definition 4: Let *R* be a relation scheme having *F* as the set of *FDs* and *JDs* over *R. R* will be in project-join normal form (*PJNF*) if for every JD *[*R1, R2*, . . ., *Rn*] which can be derived by *F* that applies to *R*, is implied by the key *FDs* of *R*.

The following example demonstrates this definition.

Example: Consider a relation scheme *R = A B C* having the set of dependencies as *F* = {A→ B C, C → A B, *[A B, B C] }. Please note that the *R* is not in *PJNF*, although since *A B* and *B C* are the super keys of *R, R* satisfies the earlier definition of *PJNF*. But R does not satisfy the revised definition as given above.

Please note that since every multivalued dependency is also a join dependency, every PJNF schema is also in 4NF. Decomposing a relation scheme using the JDs that cause PJNF violations creates the PJNF scheme. PJNF may also be not dependency preserving.

- **Check Your Progress**

1. **Define Fully Functional Dependency.**

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

2. **What is Transitivity Rule of Armstrong’s Axioms?**

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

3. **What do you mean by Lossless Join Decomposition?**

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

4. **Define Complete Set of FD?**

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

5. **Explain Merits and Demerits of Normalization.**

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

1.7LET US SUM UP

In this chapter, we have discussed about dependencies and normalization process of database. We have explored process of functional dependency with all types. We have come to know about Inferences Rules of FDs. We have also summarized Normalization Process in detail with different Normal Forms. After completion of this chapter student can able to normalize the database into proper forms.

1.8CHECK YOUR PROGRESS:POSSIBLE ANSWERS

1. Fully Functional Dependence (FFD) is defined, as Attribute Y is FFD on attribute X, if it is FD on X and not FD on any proper subset of X. According to FFD definition Y must not be FD .on any proper subset of X.
2. Transitivity Axioms is similar to the transitivity rule in algebra. If $X \rightarrow Y$ holds and $Y \rightarrow Z$, then $X \rightarrow Z$ holds.
3. A relation is decomposed into two or more smaller relations, in a way by which we can obtain the original relation by joining the decomposed partition of relation.
4. A complete set or closure set of FDs is a set of all possible FDs that can be derived from a given set of FDs. If F is used to donate the set of FDs for relation R, then a closure of a set of FDs implied by F is denoted by F^+.
5. **Merits of Normalization:**
   - More efficient data structure.
   - Avoid redundant fields or columns.
   - More flexible data structure.
   - Better understanding of data.
   - Ensures that distinct tables exist when necessary.
   - Easier to maintain data structure.
   - Minimizes data duplication.

   Demerits of Normalization:
   - You cannot start building the database before you know what the user needs.
   - On Normalizing the relations to higher normal forms i.e. 4 NF, 5 NF the performance degrades.
   - It is very time consuming and difficult process in normalizing relations of higher degree.
   - Careless decomposition may leads to bad design of database which may leads to serious problems.

1.9 Assignments

1. Explain Armstrong's Axioms of FDs. How can we find Candidate Key using it? Explain with example.

2. What is Decomposition? Explain different types of decomposition.
3. Describe Multivalued Dependencies and Join Dependencies with proper Example.
4. Explain Project Join Normal Form With Example.

1.10 Further Reading

1. Database Management S ystems, Raghu R amakrishnan and Johannes G ehrke, McGraw Hill Publication.
2. Database System Concepts, 6th Edition, Abraham Silberschatz, Henry F. Korth, S. Sudarshan, McGraw Hill.

Unit 2: Oracle Database Architecture

2

Unit Structure

2.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this chapter is to make the students,

- To learn and understand Oracle Server and Instance Architecture
- To understand the Oracle Processes
- To understand the memory structure of oracle database.
- To learn different storage structures.
- To learn schema and schema objects.

Outcome:

At the end of this unit,

- Students will be completely aware with Architecture of Oracle Database in detail.
- Students will come to know the background process and its role.
- Students will be able to simplify the different storage structures available in oracle.
- Students will be able to simplify the different schema objects available in oracle.

2.2 INTRODUCTION

Oracle Server is a database management system that provides and open, comprehensive and integrated approach to information management. In general, an Oracle server must reliably manage a large amount of data in multi user environment so that many users can concurrently access the same data. All this must be accomplished while delivering high performance. An Oracle Server must also prevent unauthorized access and provide efficient solution for failure recovery. The architecture includes physical components, memory components, processes, and logical structures.

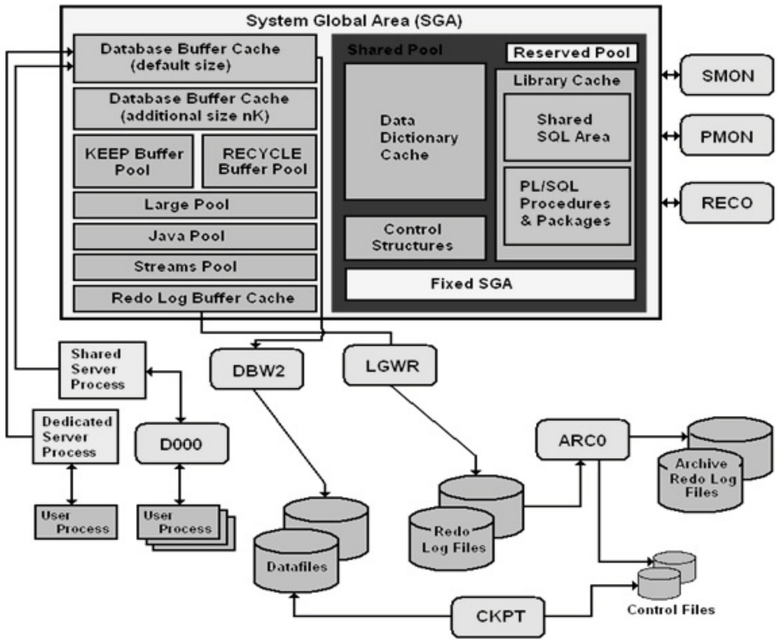


Figure 6.1: Complete Oracle Architecture

A. Oracle Server: An Oracle server includes an Oracle Instance and an Oracle database. You'll notice that the database includes several different types of files: data files, control files, redo log files and archive redo log files. The Oracle server also access parameter files and password files. This set of files has several purposes as follows:

- One is to enable system users to process SQL statements.
- Another is to improve system performance.
- Still another is to ensure the database can be recovered if there is a software/hardware failure.

B. Oracle Instance: An Oracle Instance consists of two different sets of components. The first component set is the set of background processes likes SMON, PMON, DBW0/DBWR, RECO, LGWR, CKPT, D000 and others etc. Basically each background process is a computer program. These processes perform input/output and monitor other Oracle processes to provide good performance and database reliability.

The second component set includes the memory structures that comprise the Oracle instance. When an instance starts up, a memory structure called the System Global Area (SGA) is allocated. At this point the background processes also start. The Oracle Instance provides access to an Oracle database. An Oracle Instance opens one and only one database.

C. **Oracle Database:** An Oracle database consists of file sometimes these are referred to as operating system files, but they are actually database files that store the database information that a firm or organization needs in order to operate.

When a user connects to an Oracle server, this is termed a session. The session starts when the Oracle server validates the user for connection. The session ends when the user logs out (disconnects) or if the connection terminates abnormally (network failure or client computer failure). A user can typically have more than one concurrent session. The limit of concurrent session connections is controlled by the DBA. This connection enables users to execute SQL statements. A one-to-one correspondence between the User and Server Processes is called a Dedicated Server connection. An alternative configuration is to use a Shared Server where more than one User Process shares a Server Process.

2.3 DATABASE STRUCTURES

Each running Oracle database is associated with an Oracle Instance. When a database is started on a database server, the Oracle allocated a shred memory area called the System Global Area (SGA) and starts several Background processes. This combination of SGA and Oracle Processes is called an Oracle Instance.

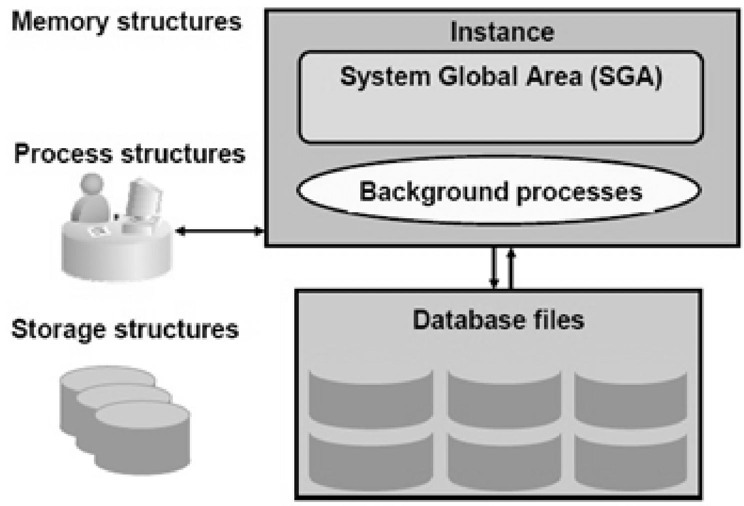


Figure 6.2: Basic Structure of Database

After Starting an instance, the Oracle associates the instance with a specific database. This is called mounting the database. The database is then ready to be opened, which makes it accessible to authorized users. Multiple instances can execute concurrently on the same computer, each accessing its own physical database.

2.4 ORACLE MEMORY STRUCTURES

The basic memory structures associated with an Oracle Instance include the following:

- **System Global Area (SGA):** Shared by all the server and background processes.
- **Program Global Area (PGA):** Private to each server and background processes. There is one PGA for each process.

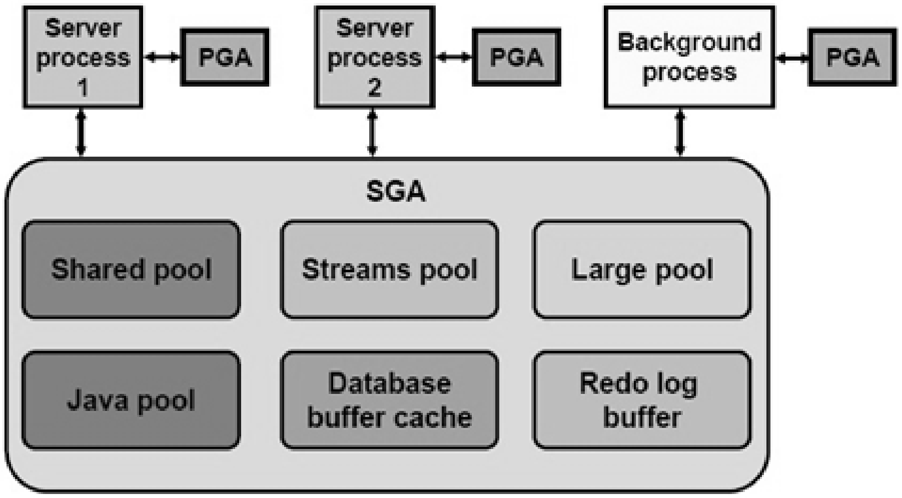


Figure 6.3: Memory Structure

2.4.1 SYSTEM GLOBAL AREA (SGA)

The System Global Area (SGA) is a memory area that contains data and control information for the instance. This information includes both organizational data and control information used by the Oracle Server. The size of the SGA is established by the parameter SGA_MAX_SIZE in the parameter file. The SGA is allocated when an Oracle instance is started up based on values specified in the initialization parameter file.

The SGA has the following mandatory memory structures:

- Shared Pool (Includes two Components)
    - Library Catch
    - Data Dictionary Cache
- Database Buffer Cache
- Redo Log Buffer
- Other structures (for example, lock and latch management, statistical data)

Additional optional memory structures in the SGA include:

- Large Pool
- Java Pool

- Streams Pool

Earlier versions of the Oracle Server used a Static SGA. This meant that if modifications to memory management were required, the database had to be shutdown, modifications were made to the init.ora parameter file, and then the database had to be restarted. After oracle 9i its use a Dynamic SGA. Memory configurations for the system global area can be made without shutting down the database instance.

Several initialization parameters are set that affect the amount of random access memory dedicated to the SGA of an Oracle Instance as follows:

- **SGA_MAX_SIZE:** This sets a limit on the amount of virtual memory allocated to the SGA – a typical setting might be 1GB; however, if the value for SGA_MAX_SIZE in the initialization parameter file or server parameter file is less than the sum the memory allocated for all components, either explicitly in the parameter file or by default, at the time the instance is initialized, then the database ignores the setting for SGA_MAX_SIZE.
- **DB_CACHE_SIZE:** This is the size of the Database Buffer Cache in standard database blocks. Block sizes vary among operating systems. We use 8KB block sizes. The total blocks in the cache defaults to 48 MB on LINUX/UNIX and 52 MB on Windows operating systems.
- **LOG_BUFFER:** This is the number of bytes allocated for the Redo Log Buffer.
- **SHARED_POOL_SIZE:** This is the number of bytes of memory allocated to shared SQL and PL/SQL. The default is 16 MB. If the operating system is based on a 64 bit configuration, then the default size is 64 MB.
- **LARGE_POOL_SIZE:** Since this is an optional memory object, the size of the Large Pool defaults to zero. If the init.ora parameter PARALLEL_AUTOMATIC_TUNING is set to TRUE, then the default size is automatically calculated.

- **JAVA_POOL_SIZE:** This is another optional memory object. The default is 24 MB of memory.

The size of the SGA cannot exceed the parameter SGA_MAX_SIZE minus the combination of the size of the additional parameters, DB_CACHE_SIZE, LOG_BUFFER, SHARED_POOL_SIZE, LARGE_POOL_SIZE, and JAVA_POOL_SIZE.

A. Shared Pool

The Shared Pool is a memory structure that is shared by all system users. It consists of both fixed and variable structures. The variable component grows and shrinks depending on the demands placed on memory size by system users and application programs. It Includes Library Cache and Data Dictionary Cache.

Memory is allocated to the Shared Pool by the parameter SHARED_POOL_SIZE in the parameter file. You can alter the size of the shared pool dynamically with the ALTER SYSTEM SET command. You must keep in mind that the total memory allocated to the SGA is set by the SGA_MAX_SIZE parameter and since the Shared Pool is part of the SGA, you cannot exceed the maximum size of the SGA.

The Shared Pool stores the most recently executed SQL statements and used data definitions. This is because some system users and application programs will tend to execute the same SQL statements often.

I. Library Cache

Memory is allocated to the Library Cache whenever an SQL statement is parsed or a program unit is called. This enables storage of the most recently used SQL and PL/SQL statements. If the Library Cache is too small, the Library Cache must purge statement definitions in order to have space to load new SQL and PL/SQL statements. Actual management of this memory structure is through a Least-Recently-Used (LRU) algorithm. This means that the SQL and PL/SQL statements that are oldest and least recently used are purged when more storage space is needed.

The Library Cache is composed of two memory subcomponents:

- **Shared SQL:** This stores/shares the execution plan and parse tree for SQL statements. If a system user executes an identical statement, then the statement does not have to be parsed again in order to execute the statement.
- **Shared PL/SQL:** Procedures and Packages: This stores/shares the most recently used PL/SQL statements such as functions, packages, and triggers.

II. Data Dictionary Cache

The Data Dictionary Cache is a memory structure that caches data dictionary information that has been recently used. This includes user account information, data file names, table descriptions, user privileges, and other information.

The database server manages the size of the Data Dictionary Cache internally and the size depends on the size of the Shared Pool in which the Data Dictionary Cache resides. If the size is too small, then the data dictionary tables that reside on disk must be queried often for information and this will slow down performance.

B. Database Buffer Cache

The Database Buffer Cache is a fairly large memory object that stores the actual data blocks that are retrieved from data files by system queries and other data manipulation language commands. A query causes a Server Process to first look in the Database Buffer Cache to determine if the requested information happens to already be located in memory – thus the information would not need to be retrieved from disk and this would speed up performance. If the information is not in the

Database Buffer Cache, the Server Process retrieves the information from disk and stores it to the cache.

Keep in mind that information read from disk is read a block at a time, not a row at a time, because a database block is the smallest addressable storage space on disk. Database blocks are kept in the Database Buffer Cache according to a Least Recently Used (LRU) algorithm and are aged out of memory if a buffer cache block is not used in order to provide space for the insertion of newly needed database blocks.

The buffers in the cache are organized in two lists:

- **Write List:** The write list holds dirty buffers – these are buffers that hold that data that has been modified, but the blocks have not been written back to disk.

- **Least Recently Used (LRU) List:** The LRU list holds free buffers, pinned buffers, and dirty buffers that have not yet been moved to the write list. Free buffers do not contain any useful data and are available for use. Pinned buffers are currently being accessed.

When an Oracle process accesses a buffer, the process moves the buffer to the most recently used (MRU) end of the LRU list – this causes dirty buffers to age toward the LRU end of the LRU list.

When an Oracle user process needs a data row, it searches for the data in the database buffer cache because memory can be searched more quickly than hard disk can be accessed. If the data row is already in the cache (a cache hit), the process reads the data from memory; otherwise a cache miss occurs and data must be read from hard disk into the database buffer cache.

Before reading a data block into the cache, the process must first find a free buffer. The process searches the LRU list, starting at the LRU end of the list. The search continues until a free buffer is found or until the search reaches the threshold limit of buffers.

Each time the user process finds a dirty buffer as it searches the LRU, that buffer is moved to the write list and the search for a free buffer continues. When the process finds a free buffer, it reads the data block from disk into the buffer and moves the buffer to the MRU end of the LRU list.

If an Oracle user process searches the threshold limit of buffers without finding a free buffer, the process stops searching the LRU list and signals the DBW0 background process to write some of the dirty buffers to disk. This frees up some buffers.

The block size for a database is set when a database is created and is determined by the init.ora parameter file parameter named DB_BLOCK_SIZE. Typical block sizes are 2K, 4K, 8K, 16K, and 32K. The size of blocks in the Database Buffer Cache matches the block size for the database.

C. Redo Log Buffer

The Redo Log Buffer memory object stores images of all changes made to database blocks. As you know, database blocks typically store several table rows of organizational data. This means that if a single column value from one row in a block is changed, the image is stored. Changes include INSERT, UPDATE, DELETE, CREATE, ALTER, or DROP.

Think of the Redo Log Buffer as a circular buffer that is reused over and over. As the buffer fills up, copies of the images are stored to the Redo Log Files that are covered in more detail in a later module.

D. Large Pool

The Large Pool is an optional memory structure that primarily relieves the memory burden placed on the Shared Pool. The Large Pool size is set with the LARGE_POOL_SIZE parameter – this is not a dynamic parameter. It does not use an LRU list to manage memory.

E. Java Pool

The Java Pool is an optional memory object, but is required if the database has Oracle Java installed and in use for Oracle JVM. The size is set with the JAVA_POOL_SIZE parameter that defaults to 24MB. The Java Pool is used for memory allocation to parse Java commands. Storing Java code and data in the Java Pool is analogous to SQL and PL/SQL code cached in the Shared Pool.

F. Streams Pool

It is sized with the parameter STREAMS_POOL_SIZE. This pool stores data and control structures to support the Oracle Streams. Oracle Steams manages sharing of data and events in a distributed environment.

2.4.2 PROGRAM GLOBAL AREA (PGA)

The Program Global Area (PGA) is also termed the Process Global Area (PGA) and is a part of memory allocated that is outside of the Oracle Instance. The PGA stores data and control information for a single Server Process or a single Background Process. It is allocated when a process is created and the memory is scavenged by the operating system when the process terminates. This is NOT a shared part of memory – one PGA to each process only.
The content of the PGA varies, but generally includes the following:

- **Private SQL Area:** Data for binding variables and runtime memory allocations. A user session issuing SQL statements has a Private SQL Area that may be associated with a Shared SQL Area if the same SQL statement is being executed by more than one system user. This often happens in OLTP environments where many users are executing and using the same application program.
  - **Dedicated Server environment:** the Private SQL Area is located in the Program Global Area.
  - **Shared Server environment:** the Private SQL Area is located in the System Global Area.
- **Session Memory:** Memory that holds session variables and other session information.
- **Software Code Area:** Software code areas store Oracle executable files running as part of the Oracle instance. These code areas are static in nature and are located in privileged memory that is separate from other user programs. The code can be installed sharable when multiple Oracle instances execute on the same server with the same software release level.

2.5 PROCESS STRUCTURE

When you invoke an application program or an Oracle tool, such as Enterprise Manager, the Oracle server creates a server process to execute the commands issued by the application. The Oracle server also creates a set of background processes for an instance hat interact with each other and with the operating system to manage the memory structures asynchronously perform I/O to write data to disk, and perform other required tasks. Which background processes a represent depends on the features that are being used in the database.

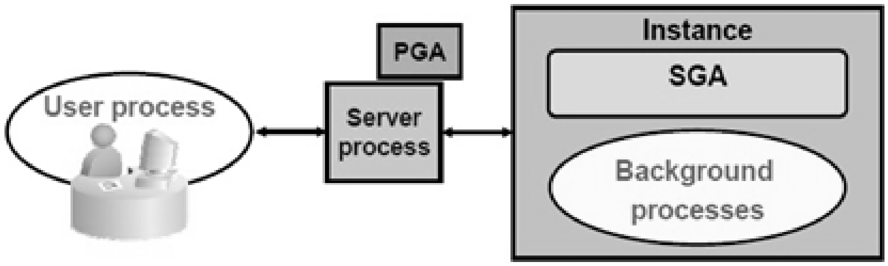


Figure 6.4: Process Structure

Process Structure includes mainly three processes as follows:

- **User Process:** When a database user requests a connection to the Oracle Server it's started.
- **Server Process:** When user established a session and connects with oracle instance it will be started.
- **Background Process:** When Oracle Instance is stated then background process will started.

A. User Process

In order to use Oracle, you must obviously connect to the database. This must occur whether you're using SQL*Plus, an Oracle tool such as Designer or Forms, or an application program.

This generates a User Process that generates programmatic calls through your user interface that creates a session and causes the generation of a Server Process that is either dedicated or shared.

B. Server Process

The Server Process is the go-between for a User Process and the Oracle Instance. In a Dedicated Server environment, there is a single Server Process to serve each User Process. In a Shared Server environment, a Server Process can serve several User Processes, although with some performance reduction.

C. Background Processes

As is shown here, there are both mandatory and optional background processes that are started whenever an Oracle Instance starts up. These background processes serve all system users. We will cover mandatory process in detail.

Mandatory background processes:

- DBWn PMON CKPT
- LGWR SMON

Optional background processes:

- ARCn LMDn RECO
- CJQ0 LMON Snnn
- Dnnn Pnnn
- LCKn QMNn

Figure 6.5: Oracle Background Process

a. **Database Writer (DBWn / DBWR):** The Database Writer writes modified blocks from the database buffer cache to the datafiles. Although one database writer process (DBW0) is sufficient for most systems, you can configure up to 20 DBWn processes (DBW0 through DBW9 and DBWa through DBWj) in order to improve write performance for a system that modifies data heavily. The initialization parameter DB_WRITER_PROCESSES specifies the number of DBWn processes.

   The purpose of DBWn is to improve system performance by caching writes of database blocks from the Database Buffer Cache back to datafiles. Blocks that have been modified and that need to be written back to disk are termed "dirty blocks." The DBWn also ensures that there are enough free buffers in the Database Buffer Cache to service Server Processes that may be reading data from datafiles into the Database Buffer Cache. Performance improves because by delaying writing changed database blocks back to disk, a Server Process may find the data that is needed to meet a User Process request already residing in memory.

b. **Log Writer (LGWR):** The Log Writer (LGWR) writes contents from the Redo Log Buffer to the Redo Log File that is in use. These are sequential writes since the Redo Log Files record database modifications based on the actual time that the modification takes place. LGWR actually writes before the DBWn writes and only confirms that a COMMIT operation has succeeded when the Redo Log Buffer contents are successfully written to disk. LGWR can also call the DBWn to write contents of the Database Buffer Cache to disk.

c. **System Monitor (SMON):** The System Monitor (SMON) is responsible for instance recovery by applying entries in the online redo log files to the datafiles.

If an Oracle Instance fails, all information in memory not written to disk is lost. SMON is responsible for recovering the instance when the database is started up again. It does the following:

- Rolls forward to recover data that was recorded in a Redo Log File, but that had not yet been recorded to a datafile by DBWn. SMON reads the Redo Log Files and applies the changes to the data blocks. This recovers all transactions that were committed because these were written to the Redo Log Files prior to system failure.
- Opens the database to allow system users to logon.
- Rolls back uncommitted transactions.

SMON also does limited space management. It combines adjacent areas of free space in the database's datafiles for tablespaces that are dictionary managed. It also de-allocates temporary segments to create free space in the data files.

d. **Process Monitor (PMON):** The Process Monitor (PMON) is a cleanup type of process that cleans up after failed processes such as the dropping of a user connection due to a network failure or the abend of a user application program.

e. **Checkpoint (CKPT):** The Checkpoint (CPT) process writes information to the database control files that identifies the point in time with regard to the Redo Log Files where instance recovery is to begin should it be necessary. This is done at a minimum, once every three seconds.

Think of a checkpoint record as a starting point for recovery. DBWn will have completed writing all buffers from the Database Buffer Cache to disk prior to the checkpoint, thus those record will not require recovery. This does the following:

- Ensures modified data blocks in memory are regularly written to disk – CKPT can call the DBWn process in order to ensure this and does so when writing a checkpoint record.
- Reduces Instance Recovery time by minimizing the amount of work needed for recovery since only Redo Log File entries processed since the last checkpoint require recovery.
- Causes all committed data to be written to datafiles during database shutdown.

If a Redo Log File fills up and a switch is made to a new Redo Log File (this is covered in more detail in a later module), the CKPT process also writes checkpoint information into the headers of the datafiles.

Checkpoint information written to control files includes the system change number (the SCN is a number stored in the control file and in the headers of the database files that are used to ensure that all files in the system are synchronized), location of which Redo Log File is to be used for recovery, and other information. CKPT does not write data blocks or redo blocks to disk – it calls DBWn and LGWR as necessary.

Optional Background Process:

f. **Archiver (ARCn):** We cover the Archiver (ARCn) optional background process in more detail because it is almost always used for production systems storing mission critical information. The ARCn process must be used to recover from loss of a physical disk drive for systems that are "busy" with lots of transactions being completed.

When a Redo Log File fills up, Oracle switches to the next Redo Log File. The DBA creates several of these and the details of creating them are covered in a later module. If all Redo Log Files fill up, then Oracle switches back to the first one and uses them in a round-robin fashion by overwriting ones that have already been used – it should be obvious that the information stored on the files, once overwritten, is lost forever. If ARCn is in what is termed ARCHIVELOG mode, then as the Redo Log Files fill up, they are individually written to Archived Redo Log Files and LGWR does not overwrite a Redo Log File until archiving has completed. Thus, committed data is not lost forever and can be recovered in the event of a disk failure. Only the contents of the SGA will be lost if an Instance fails.

In NOARCHIVELOG mode, the Redo Log Files are overwritten and not archived. Recovery can only be made to the last full backup of the database files.

When running in ARCHIVELOG mode, the DBA is responsible to ensure that the Archived Redo Log Files do not consume all available disk space! Usually after two complete backups are made, any Archived Redo Log Files for prior backups are deleted.

g. **Coordinator Job Queue (CJQ0):** Coordinator Job Queue – This is the coordinator of job queue processes for an instance. It monitors the JOB$ table (table of jobs in the job queue) and starts job queue processes (Jnnn) as needed to execute jobs The Jnnn processes execute job requests created by the DBMS_JOBS package.

h. **Dispatcher Process (Dnnn):** Dispatcher number "nnn", for example, D000 would be the first dispatcher process – Dispatchers are optional background processes, present only when the shared server configuration is used.

i. **Recovery (RECO):** The Recovery process is used to resolve distributed transactions that are pending due to a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions.

2.6 STORAGE STRUCTURE

An Oracle database consists of file sometimes these are referred to as operating system files, but they are actually database files that store the database information that a firm or organization needs in order to operate. Database Storage Structures divided into two parts as follows:

- Physical Structure
- Logical Structure

2.6.1 PHYSICAL DATABASE STRUCTURE

An Oracle database consists of physical files shown as below figure.

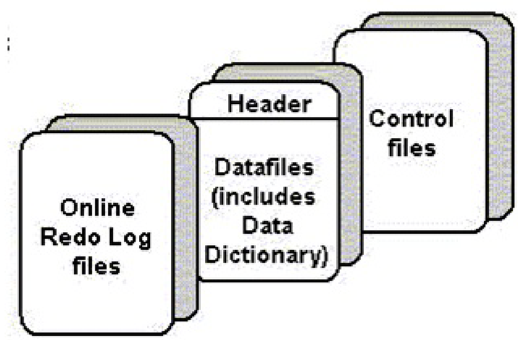


Figure 6.6: Physhical Storage Structure

The files that constitute an Oracle Database are organized into the following:

A. Control Files: Contains data about the database itself. These files are critical to database. Without it, cannot open data files to access data within the database. It is used to synchronize all database activities.

B. Data Files: Contain the actual data for the database.

C. Redo Log Files: Contain a record of changes made to the database, and allow recovery when a database failure occurs. If the database crashes and does not lose any data files, then the instance can recover the database with the information in these files.

Other key files as noted above include:

- **Parameter file:** It used to define how the instance is configured when its start up. There are two types of parameter files.
  - **The init.ora file (also called the PFILE):** is a static parameter file. It contains parameters that specify how the database instance is to start up. For example, some parameters will specify how to allocate memory to the various parts of the system global area.
  - **The spfile.ora**: is a dynamic parameter file. It also stores parameters to specify how to startup a database; however, its parameters can be modified while the database is running.
- **Password file:** Specifies which special users are authenticated to startup/shut down an Oracle Instance. Also allows user to connect remotely to the database.
- **Archived redo log files:** Contain an ongoing history of the data change generated by instance. We can say that, it is copy of the redo log files and are necessary for recovery in an online, transaction-processing environment in the event of a disk failure.
- **Backup files:** Are used for database recovery. Typically restore a backup files when a media failure or user error has damaged or deleted the original file.
- **Trace Files:** Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to trace file is intended for the database administrator.
- **Alert Log Files:** There are special trace files. They are also known as alert logs. The alert log of a database is a chronological log of messages and errors.

2.6.2 LOGICAL STRUCTURE

It is helpful to understand how an Oracle database is organized in terms of a logical structure that is used to organize physical objects.

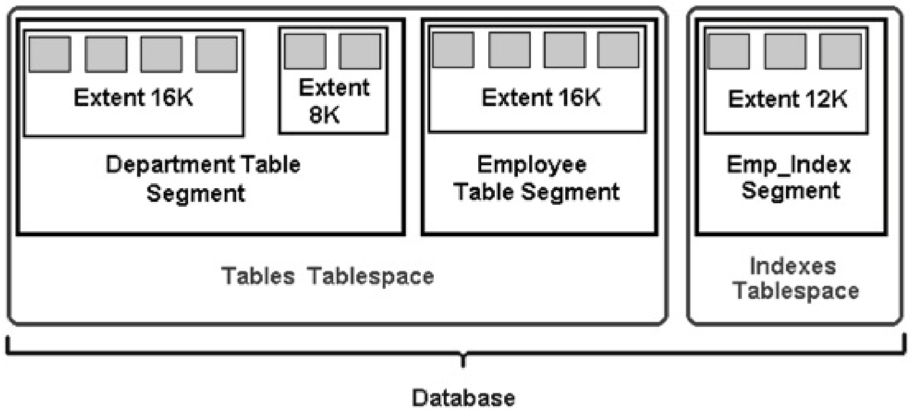


Figure 6.7: Logical Storage Structure

- **Tablespace:** An Oracle 10g database must always consist of at least two tablespaces (SYSTEM and SYSAUX), although a typical Oracle database will multiple tablespaces tablespaces. A tablespace is a logical storage facility (a logical container) for storing objects such as tables, indexes, sequences, clusters, and other database objects.

  Each tablespace has at least one physical datafile that actually stores the tablespace at the operating system level. A large tablespace may have more than one datafile allocated for storing objects assigned to that tablespace. A tablespace belongs to only one database. Tablespace can be brought online and taken offline for purposes of backup and management, except for the SYSTEM tablespace that must always be online. Tablespaces can be in either read-only or read-write status.
- **Datafile:** Tablespaces are stored in datafiles which are physical disk objects. A datafile can only store objects for a single tablespace, but a tablespace may have more than one datafile – this happens when a disk drive device fills up and a tablespace needs to be expanded, then it is expanded to a new disk drive. The DBA can change the size of a datafile to make it smaller or later. The file can also grow in size dynamically as the tablespace grows.

- **Segment:** When logical storage objects are created within a tablespace, for example, an employee table, a segment is allocated to the object. Obviously a tablespace typically has many segments. A segment cannot span tablespaces but can span datafiles that belong to a single tablespace.

- **Extent:** Each object has one segment which is a physical collection of extents. Extents are simply collections of contiguous disk storage blocks. A logical storage object such as a table or index always consists of at least one extent – ideally the initial extent allocated to an object will be large enough to store all data that is initially loaded. As a table or index grows, additional extents are added to the segment. A DBA can add extents to segments in order to tune performance of the system. An extent cannot span a datafile.

- **Data Block:** The Oracle Server manages data at the smallest unit in what is termed a block or data block. Data are actually stored in blocks.

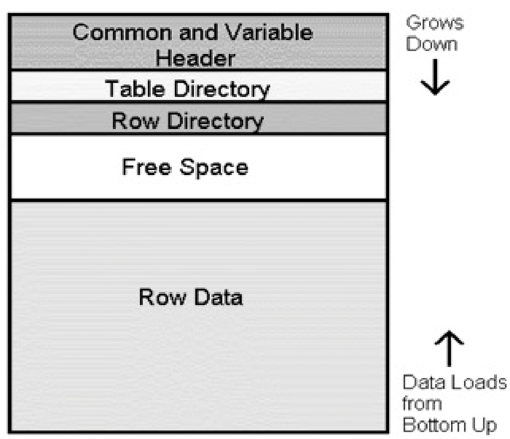


Figure 6.8: Structure of Data Block

A physical block is the smallest addressable location on a disk drive for read/write operations. An Oracle data block consists of one or more physical blocks (operating system blocks) so the data block, if larger than an operating system block, should be an even multiple of the operating system block size, e.g., if the UNIX operating system block size is 2K or 4K, then the Oracle data block should be 2K, 4K, 8K, 12K, 16K, etc in size. This optimizes I/O.

The data block size is set at the time the database is created and cannot be changed. It is set with the DB_BLOCK_SIZE parameter. The maximum data block size depends on the operating system.

2.7 SCHEMA AND SCHEMA OBJECTS

A schema is a collection of database objects. A schema is owned by a database user and has the same name as that user. A schema is a collection of schema objects.

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

Different types of objects contained in a user's schema. It includes:

- **Tables:** Tables are the basic unit of data storage in an Oracle database. Data is stored in rows and columns.
- **Views:**A view is a tailored presentation of the data contained in one or more tables. A view takes the output of a query and treats it as a table; therefore, also known as virtual table.
- **Synonyms:**A synonym is an alias for any table, view, snapshot, sequence, procedure, function, or package. Because a synonym is simply an alias, it requires no storage.
- **Indexes:**Indexes are optional structures associated with tables and clusters. It is used to speed SQL statement execution on a table.
- **Clusters:** A cluster is a group of tables that share the same data blocks because they share common columns and are often used together.
- **Hash Clusters:**A hash cluster stores related rows together in the same data blocks. Rows in a hash cluster are stored together based on their hash value.

➢ **Check Your Progress**

1. **List Components of Oracle Instance?**

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

2. **Which Parameter is used to define size of SGA? Maximum size of SGA Is?**

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

3. **Which Background Process is Responsible for Instance Recovery?**

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

4. **Explain Archived Redo Log File?**

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

5. **Is there more than One Data files in a single Tablespace?**

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

2.8 LET US SUM UP

In this chapter, we have discussed about oracle architecture and instance. We have also explored memory structure of Oracle Database. We have come to know vital processes, which is executes during database execution. We have also summarized storage structures and supported files and architectures. After completion of this chapter we came to know about schemas and various schema objects.

2.9 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Oracle Instance consists of Two components namely Memory Structure and Background Processes.
2. SGA_MAX_SIZE parameter of Initialization Parameter file is used to define size of SGA. The size of the SGA cannot exceed the parameter SGA_MAX_SIZE minus the combination of the size of the additional parameters, DB_CACHE_SIZE, LOG_BUFFER, SHARED_POOL_SIZE, LARGE_POOL_SIZE, and JAVA_POOL_SIZE.
3. System Monitor (SMON) is responsible for instance recovery by applying entries in the online redo log files to the datafiles.
4. Archived Redo Log File is the copy of redo log files and necessary for recovery in the event of disk failure.
5. Yes, A Large tablespace may have more than one datafiles.

2.10 ASSIGNMENTS

1. Explain SGA in detail.
2. What is Database Buffer Cache? Explain in detail with parameters.
3. Describe all Background Processes.
4. Explain Logical Database Storage Structures.
5. Define Schema and Schema Objects in detail.

2.11 FURTHER READING

1. Expert Oracle Database Architecture, Third Edition, Darl Kuhn & Thomas Kyte, Apress Publishing.
2. Oracle Database 10g The Complete Reference, Kevin Loney, Oracle Press.
3. Advanced RDBMS Using Oracle, Himanshu Dabir & Dipali Mehar, Vision Publication.

Unit 3: Distributed Database Architecture

3

Unit Structure

3.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this chapter is,

• To learn and understand Different Distributed Database Architectures

• To understand the Client/Server Database Architecture.

• To understand Database Links and Users.

• To learn security aspects into the Distributed Database Environment.

• To learn Distributed Database Application Development

Outcome:

At the end of this unit,

• Students will be completely aware with Homogenous and Heterogeneous Distributed Architectures.

• Students will come to know about different types of Database Links and Restrictions of Database Links.

• Students will be able to simplify the Remote Procedure Call (RPC) Mechanism.

• Students will be able to simplify Query Optimization in Distributed Environments.

3.2 INTRODUCTION

A distributed database system allows applications to access data from local and remote databases. In a homogenous distributed database system, each database is an Oracle Database. In a heterogeneous distributed database system, at least one of the databases is not an Oracle Database. Distributed databases use client/server architecture to process information requests. In this chapter will learn different concepts as follows:

- Homogenous Distributed Database Systems
- Heterogeneous Distributed Database Systems
- Client/Server Database Architecture
- Database Links

- Database Security Aspects
- Distributed Query Optimization

3.3 Homogenous Distributed Database Systems

A homogenous distributed database system is a network of two or more Oracle Databases that reside on one or more systems. An application can simultaneously access or modify the data in several databases in a single distributed environment.

You can also create synonyms for remote objects in the distributed system so that users can access them with the same syntax as local objects. In this way, a distributed system gives the appearance of native data access. Users on mfg do not have to know that the data they access resides on remote databases.

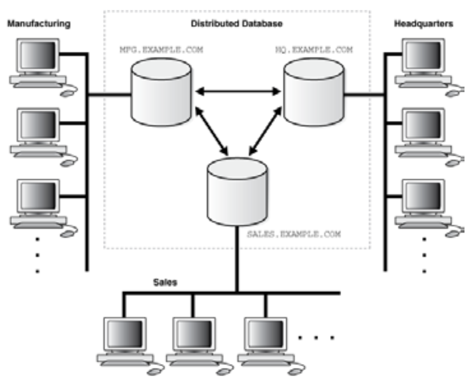


Figure 7.1: Homogenous Distributed Database Systems

An Oracle Database distributed database system can incorporate Oracle Databases of different versions. All supported releases of Oracle Database can participate in a distributed database system. Nevertheless, the applications that work with the distributed database must understand the functionality that is available at each node in the system. A distributed database application cannot expect an Oracle7 database to understand the SQL extensions that are only available with Oracle Database.

I. Distributed Databases Vs Distributed Processing

The terms distributed database and distributed processing are closely related, yet have distinct meanings. There definitions are as follows:

- **Distributed database:** A set of databases in a distributed system that can appear to applications as a single data source.

- **Distributed processing:** the operation that occurs when an application distributes its tasks among different computers in a network. For example, a database application typically distributes front-end presentation tasks to client computers and allows a back-end database server to manage shared access to a database. Consequently, a distributed database application processing system is more commonly referred to as a client/server database application system.

Distributed database systems employ a distributed processing architecture. For example, an Oracle Database server acts as a client when it requests data that another Oracle Database server manages.

3.4 Heterogeneous Distributed Database System

In a heterogeneous distributed database system, at least one of the databases is a non-Oracle Database system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle Database. The local Oracle Database server hides the distribution and heterogeneity of the data.

The Oracle Database server accesses the non-Oracle Database system using Oracle Heterogeneous Services with an agent. If you access the non-Oracle Database data store using an Oracle Transparent Gateway, then the agent is a system-specific application. For example, if you include a Sybase database in an Oracle Database distributed system, then you must obtain a Sybase-specific transparent gateway so that the Oracle Database in the system can communicate with it.

Alternatively, you can use generic connectivity to access non-Oracle Database data stores so long as the non-Oracle Database system supports the ODBC or OLE DB protocols.

A. Heterogeneous Services

Heterogeneous Services (HS) is an integrated component within the Oracle Database server and the enabling technology for the current suite of Oracle Transparent Gateway products. HS provides the common architecture and administration mechanisms for Oracle Database gateway products and other heterogeneous access facilities. Also, it provides upwardly compatible functionality for users of most of the earlier Oracle Transparent Gateway releases.

B. Transparent Gateway Agents

For each non-Oracle Database system that you access, Heterogeneous Services can use a transparent gateway agent to interface with the specified non-Oracle Database system. The agent is specific to the non-Oracle Database system, so each type of system requires a different agent.

The transparent gateway agent facilitates communication between Oracle Database and non-Oracle Database systems and uses the Heterogeneous Services component in the Oracle Database server. The agent executes SQL and transactional requests at the non-Oracle Database system on behalf of the Oracle Database server.

C. Generic Connectivity

Generic connectivity enables you to connect to non-Oracle Database data stores by using either a Heterogeneous Services ODBC agent or a Heterogeneous Services OLE DB agent. Both are included with your Oracle product as a standard feature. Any data source compatible with the ODBC or OLE DB standards can be accessed using a generic connectivity agent.

The advantage to generic connectivity is that it may not be required for you to purchase and configure a separate system-specific agent. You use an ODBC or OLE DB driver that can interface with the agent. However, some data access features are only available with transparent gateway agents.

3.5 CLIENT/SERVER DATABASE ARCHITECTURE

A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

In Figure 7-2, the host for the hq database is acting as a database server when a statement is issued against its local data (for example, the second statement in each transaction issues a statement against the local dept table), but is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table emp in the sales database).

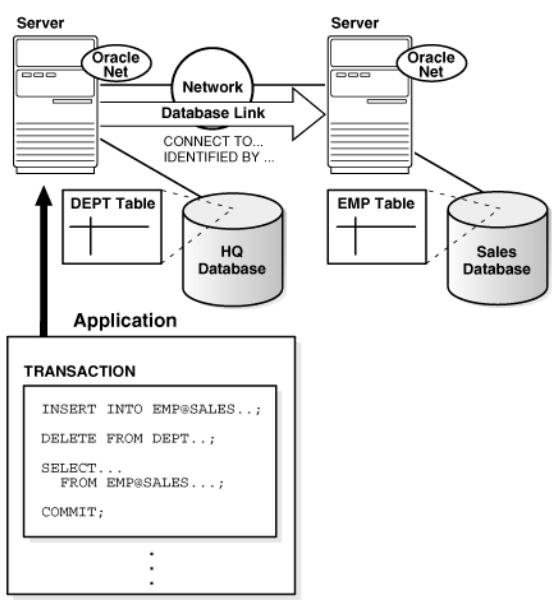


Figure 7.2: An Oracle Database Distributed Database System

A client can connect directly or indirectly to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server.

3.6 DATABASE LINKS

The central concept in distributed database systems is a database link. A database link is a connection between two physical database servers that allows a client to access them as one logical database.

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 is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B.

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.

Database links are either private or public. If they are private, then only the user who created the link has access; if they are public, then all database users have access. One principal difference among database links is the way that connections to a remote database occur. Users access a remote database through the following types of links:

Type of Link	Description

Type of Link	Description
Connected user link	Users connect as themselves, which means that they must have an account on the remote database with the same user name and password as their account on the local database.
Fixed user link	Users connect using the username and password referenced in the link.
Current user link	A user connects as a global user. A local user can connect as a global user in the context of a stored procedure, without storing the global user's password in a link definition.

Create database links using the CREATE DATABASE LINK statement. After a link is created, you can use it to specify schema objects in SQL statements.

3.6.1 SHARED DATABASE LINKS

A shared database link is a link between a local server process and the remote database. The link is shared because multiple client processes can use the same link simultaneously.

When a local database is connected to a remote database through a database link, either database can run in dedicated or shared server mode. The following table illustrates the possibilities:

Local Database Mode	Remote Database Mode
Dedicated	Dedicated
Dedicated	Shared server
Shared server	Dedicated
Shared server	Shared server

A shared database link can exist in any of these four configurations. Shared links differ from standard database links in the following ways:

- Different users accessing the same schema object through a database link can share a network connection.
- When a user must establish a connection to a remote server from a particular server process, the process can reuse connections already established to the remote server. The reuse of the connection can occur if the connection was established on the same server process with the same database link, possibly in a different session. In a non-shared database link, a connection is not shared across multiple sessions.
- When you use a shared database link in a shared server configuration, a network connection is established directly out of the shared server process in the local server. For a non-shared database link on a local shared server, this connection would have been established through the local dispatcher, requiring context switches for the local dispatcher, and requiring data to go through the dispatcher.

The great advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object owner. In other words, a local user can access a link to a remote database without having to be a user on the remote database.

3.6.2. TYPES OF DATABASE LINKS

Oracle Database lets you create private, public, and global database links. These basic link types differ according to which users are allowed access to the remote database:

Type	Owner	Description
Private	User who created the link. View ownership data through: • DBA_DB_LINKS	Creates link in a specific schema of the local database. Only the owner of a private database link or PL/SQL subprograms in

Type	Owner	Description
	• ALL_DB_LINKS • USER_DB_LINKS	the schema can use this link to access database objects in the corresponding remote database.
Public	User called PUBLIC. View ownership data through views shown for private database links.	Creates a database-wide link. All users and PL/SQL subprograms in the database can use the link to access database objects in the corresponding remote database.
Global	User called PUBLIC. View ownership data through views shown for private database links.	Creates a network-wide link. When an Oracle network uses a directory server, the directory server automatically create and manages global database links (as net service names) for every Oracle Database in the network. Users and PL/SQL subprograms in any database can use a global link to access objects in the corresponding remote database. Note: In earlier releases of Oracle Database, a global database link referred to a database link that was registered with an Oracle Names server. The use of an Oracle Names server has been deprecated. In this document, global database links refer to the use of net service names from the directory server.

Determining the type of database links to employ in a distributed database depends on the specific requirements of the applications using the system. Consider these features when making your choice:

Type of Link	Features
Private database link	This link is more secure than a public or global link, because only the owner of the private link, or subprograms within the same schema, can use the link to access the remote database.
Public database link	When many users require an access path to a remote Oracle Database, you can create a single public database link for all users in a database.
Global database link	When an Oracle network uses a directory server, an administrator can conveniently manage global database links for all databases in the system. Database link management is centralized and simple.

3.6.3. USERS OF DATABASE LINKS

When creating the link, you determine which user should connect to the remote database to access the data. The following table explains the differences among the categories of users involved in database links:

User Type	Description
Connected user	A local user accessing a database link in which no fixed username and password have been specified. If SYSTEM accesses a public link in a query, then the connected user is SYSTEM, and the database connects to the SYSTEM schema in the remote database. Note: A connected user does not have to be the user who created the link, but is any user who is accessing the link.
Current user	A global user in a CURRENT_USER database link. The global user must be authenticated by an X.509 certificate (an SSL-authenticated enterprise user) or a password (a password-

User Type	Description
	authenticated enterprise user), and be a user on both databases involved in the link. Current user links are an aspect of the Oracle Advanced Security option. See Oracle Database Advanced Security Administrator's Guide for information about global security
Fixed user	A user whose username/password is part of the link definition. If a link includes a fixed user, the fixed user's username and password are used to connect to the remote database.

3.6.4. DATABASE LINK RESTRICTIONS

You cannot perform the following operations using database links:

- Grant privileges on remote objects
- Execute DESCRIBE operations on some remote objects. The following remote objects, however, do support DESCRIBE operations:
    - Tables
    - Views
    - Procedures
    - Functions
- Analyze remote objects
- Define or enforce referential integrity
- Grant roles to users in a remote database
- Obtain non-default roles on a remote database.
- Execute hash query joins that use shared server connections

3.7 DISTRIBUTED DATABASE SECURITY

The database supports all of the security features that are available with a non-distributed database environment for distributed database systems, including:

- Password authentication for users and roles
- Some types of external authentication for users and roles including Kerberos version 5 for connected user links.
- Login packet encryption for client-to-server and server-to-server connections

Some important concepts to consider when configuring an Oracle Database distributed database system:

- Authentication Through Database Links
- Authentication Without Passwords
- Supporting User Accounts and Roles
- Centralized User and Privilege Management
- Database Encryption

A. Authentication Through Database Links

Database links are either private or public, authenticated or non-authenticated. You create public links by specifying the PUBLIC keyword in the link creation statement. You create authenticated links by specifying the CONNECT TO clause, AUTHENTICATED BY clause, or both clauses together in the database link creation statement. For example, you can issue:

B. Authentication Without Passwords

When using a connected user or current user database link, you can use an external authentication source such as Kerberos to obtain end-to-end security. In end-to-end authentication, credentials are passed from server to server and can be authenticated by a database server belonging to the same domain.

C. Supporting User Accounts and Roles

In a distributed database system, you must carefully plan the user accounts and roles that are necessary to support applications using the system. Note that:

- The user accounts necessary to establish server-to-server connections must be available in all databases of the distributed database system.
- The roles necessary to make available application privileges to distributed database application users must be present in all databases of the distributed database system.

As you create the database links for the nodes in a distributed database system, determine which user accounts and roles each site must support server-to-server connections that use the links.

In a distributed environment, users typically require access to many network services. When you must configure separate authentications for each user to access each network service, security administration can become unwieldy, especially for large systems.

D. Centralized User and Privilege Management

The database provides different ways for you to manage the users and privileges involved in a distributed system. For example, you have these options:

- **Enterprise user management:** You can create global users who are authenticated through SSL or by using passwords, then manage these users and their privileges in a directory through an independent enterprise directory service.
- **Network authentication service:** This common technique simplifies security management for distributed environments. You can use the Oracle Advanced Security option to enhance Oracle Net and the security of an Oracle Database distributed database system. Windows NT native authentication is an example of a non-Oracle authentication solution.

E. Database Encryption

The Oracle Advanced Security option also enables Oracle Net and related products to use network data encryption and check-summing so that data cannot be read or altered.

It protects data from unauthorized viewing by using the RSA Data Security RC4 or the Data Encryption Standard (DES) encryption algorithm.

To ensure that data has not been modified, deleted, or replayed during transmission, the security services of the Oracle Advanced Security option can generate a cryptographically secure message digest and include it with each packet sent across the network.

3.8 TRANSACTION PROCESSING IN A DISTRIBUTED SYSTEM

A transaction is a logical unit of work constituted by one or more SQL statements executed by a single user. A transaction begins with the user's first executable SQL statement and ends when it is committed or rolled back by that user.

A remote transaction contains only statements that access a single remote node. A distributed transaction contains statements that access multiple nodes.

The following sections define important concepts in transaction processing and explain how transactions access data in a distributed database:

- Remote SQL Statements
- Distributed SQL Statements
- Shared SQL for Remote and Distributed Statements
- Remote Transactions
- Distributed Transactions
- Two-Phase Commit Mechanism
- Database Link Name Resolution
- Schema Object Name Resolution

A. Remote SQL Statements

A remote query statement is a query that selects information from one or more remote tables, all of which reside at the same remote node. A remote update statement is an update that modifies data in one or more tables, all of which are located at the same remote node.

B. Distributed SQL Statements

A distributed query statement retrieves information from two or more nodes. A distributed update statement modifies data on two or more nodes. A distributed update is possible using a PL/SQL subprogram unit such as a procedure or trigger that includes two or more remote updates that access data on different nodes.

C. Shared SQL for Remote and Distributed Statements

The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement. The SQL text must match, and the referenced objects must match. If available, shared SQL areas can be used for the local and remote handling of any statement or decomposed query.

D. Remote Transactions

A remote transaction contains one or more remote statements, all of which reference a single remote node.

E. Distributed Transactions

A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.

F. Two-Phase Commit Mechanism

A database must guarantee that all statements in a transaction, distributed or non-distributed, either commit or roll back as a unit. The effects of an ongoing transaction should be invisible to all other transactions at all nodes; this transparency should be true for transactions that include any type of operation, including queries, updates, or remote procedure calls.

The general mechanisms of transaction control in a non-distributed database are discussed in the Oracle Database Concepts Concepts. In a distributed database, the database must coordinate transaction control with the same characteristics over a network and maintain data consistency, even if a network or system failure occurs.

The database two-phase commit mechanism guarantees that all database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction. A two-phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

G. Database Link Name Resolution

A global object name is an object specified using a database link. The essential components of a global object name are:

- Object name
- Database name
- Domain

Whenever a SQL statement includes a reference to a global object name, the database searches for a database link with a name that matches the database name specified in the global object name.

The database performs this operation to determine the path to the specified remote database.

The database always searches for matching database links in the following order:

1. Private database links in the schema of the user who issued the SQL statement.
2. Public database links in the local database.
3. Global database links (only if a directory server is available).

H. Schema Object Name Resolution

After the local Oracle Database connects to the specified remote database on behalf of the local user that issued the SQL statement, object resolution continues as if the remote user had issued the associated SQL statement. The first match determines the remote schema according to the following rules:

Type of Link Specified	Location of Object Resolution
A fixed user database link	Schema specified in the link creation statement
A connected user database link	Connected user's remote schema
A current user database link	Current user's schema

If the database cannot find the object, then it checks public objects of the remote database. If it cannot resolve the object, then the established remote session remains but the SQL statement cannot execute and returns an error.

3.9 DISTRIBUTED DATABASE APPLICATION DEVELOPMENT

Application development in a distributed system raises issues that are not applicable in a non-distributed system. This section contains the following topics relevant for distributed application development:

- Transparency in a Distributed Database System
- Remote Procedure Calls (RPCs)

- Distributed Query Optimization

3.9.1 TRANSPARENCY IN A DISTRIBUTED DATABASE SYSTEM

With minimal effort, you can develop applications that make an Oracle Database distributed database system transparent to users that work with the system. The goal of transparency is to make a distributed database system appear as though it is a single Oracle Database. Consequently, the system does not burden developers and users of the system with complexities that would otherwise make distributed database application development challenging and detract from user productivity.

The following sections explain more about transparency in a distributed database system.

A. **Location Transparency:** An Oracle Database distributed database system has features that allow application developers and administrators to hide the physical location of database objects from applications and users. Location transparency exists when a user can universally refer to a database object such as a table, regardless of the node to which an application connects. Location transparency has several benefits, including:

   - Access to remote data is simple, because database users do not need to know the physical location of database objects.
   - Administrators can move database objects with no impact on end-users or existing database applications.

   Typically, administrators and developers use synonyms to establish location transparency for the tables and supporting objects in an application schema.

B. **SQL and COMMIT Transparency:** The Oracle Database distributed database architecture also provides query, update, and transaction transparency. For example, standard SQL statements such as SELECT, INSERT, UPDATE, and DELETE work just as they do in a non-distributed database environment.

Additionally, applications control transactions using the standard SQL statements COMMIT, SAVEPOINT, and ROLLBACK.

C. **Replication Transparency:** The database also provide many features to transparently replicate data among the nodes of the system. For more information about Oracle Database replication features, see Oracle Database Advanced Replication.

3.9.2. REMOTE PROCEDURE CALLS (RPCS)

Developers can code PL/SQL packages and procedures to support applications that work with a distributed database. Applications can make local procedure calls to perform work at the local database and remote procedure calls (RPCs) to perform work at a remote database.

When a program calls a remote procedure, the local server passes all procedure parameters to the remote server in the call.

In order for the RPC to succeed, the called procedure must exist at the remote site, and the user being connected to must have the proper privileges to execute the procedure.

When developing packages and procedures for distributed database systems, developers must code with an understanding of what program units should do at remote locations, and how to return the results to a calling application.

3.9.3 DISTRIBUTED QUERY OPTIMIZATION

Distributed query optimization is an Oracle Database feature that reduces the amount of data transfer required between sites when a transaction retrieves data from remote tables referenced in a distributed SQL statement.

Distributed query optimization uses cost-based optimization to find or generate SQL expressions that extract only the necessary data from remote tables, process that data at a remote site or sometimes at the local site, and send the results to the local site for final processing. This operation reduces the amount of required data transfer when compared to the time it takes to transfer all the table data to the local site for processing.

Using various cost-based optimizer hints such as DRIVING_SITE, NO_MERGE, and INDEX, you can control where Oracle Database processes the data and how it accesses the data.

- **Check Your Progress**

6. Define Distributed Database and Distributed Processing?

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

7. What is Generic Connectivity in Heterogeneous Distributed Database?

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

8. What is Database Links? Explain different types of Database Links.

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

9. Explain Distributed Query Optimization.

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

3.10 LET US SUM UP

In this chapter, we have discussed about oracle architecture and instance. We have also explored memory structure of Oracle Database. We have come to know vital processes, which is executes during database execution. We have also summarized storage structures and supported files and architectures. After completion of this chapter we came to know about schemas and various schema objects.

3.11 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Distributed database is a set of databases in a distributed system that can appear to applications as a single data source. While distributed processing is the operation that occurs when an application distributes its tasks among different computers in a network.
2. Generic connectivity enables you to connect to non-Oracle Database data stores by using either a Heterogeneous Services ODBC agent or a Heterogeneous Services OLE DB agent. The advantage to generic connectivity is that it may not be required for you to purchase and configure a separate system-specific agent. You use an ODBC or OLE DB driver that can interface with the agent.
3. A database link is a connection between two physical database servers that allows a client to access them as one logical database. These basic link types differ according to which users are allowed access to the remote database:

Type	Description
Private	Creates link in a specific schema of the local database. Only the owner of a private database link or PL/SQL subprograms in the schema can use this link to access database objects in the corresponding remote database.
Public	Creates a database-wide link. All users and PL/SQL subprograms in the database can use the link to access database objects in the

Type	Description
	corresponding remote database.
Global	Creates a network-wide link. When an Oracle network uses a directory server, the directory server automatically create and manages global database links (as net service names) for every Oracle Database in the network. Users and PL/SQL subprograms in any database can use a global link to access objects in the corresponding remote database.

4. Distributed query optimization is an Oracle Database feature that reduces the amount of data transfer required between sites when a transaction retrieves data from remote tables referenced in a distributed SQL statement.

3.12ASSIGNMENTS

1. Explain Homogenous and Heterogeneous Distributed Database.
2. Explain Transaction Processing in Distributed Database.
3. Describe Security Aspects in Distributed Database.
4. What is Database Links? Describe different users of Database Links in details.

3.13 Further Reading

1. Expert Oracle Database Architecture, Third Edition, Darl Kuhn & Thomas Kyte, Apress Publishing.
2. Oracle Database 10g The Complete Reference, Kevin Loney, Oracle Press.

Unit 4: Database Backup

4

Unit Structure

4.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this chapter is to make the students,

• To understand Types of Oracle Backups

• To understand the Logical Backup Plan (Export/Import)

• To understand the Physical Backup & Recovery

Outcome:

At the end of this unit,

• Students will be completely aware with Logical and Physical Backup Strategies of Oracle database.

• Students will able to Perform Export/Import with its different parameter.

• Students will be aware with different mode of Online and Offline Backup.

• Students will be aware with how to make database ready for physical database backup.

4.2 INTRODUCTION

A backup is a representative copy of data. This copy can include important parts of a database such as the control file, redo logs, and datafiles. A backup protects data from application error and acts as a safeguard against unexpected data loss, by providing a way to restore original data.

Backups are divided into physical backups and logical backups. Physical backups are copies of physical database files. The phrase "backup and recovery" usually refers to the transfer of copied files from one location to another, along with the various operations performed on these files.

In contrast, logical backups contain data that is exported using SQL commands and stored in a binary file. Oracle records both committed and uncommitted changes in redo log buffers. Logical backups are used to supplement physical backups.

Restoring a physical backup means reconstructing it and making it available to the Oracle server. To recover a restored backup, data is updated using redo records from the transaction log. The transaction log records changes made to the database after the backup was taken.

Elements of a Backup And Recovery Strategy

Although backup and recovery operations can be intricate and vary from one business to another, the basic principles follow these four simple steps:

1. Multiplex the online redo logs
2. Run the database in ARCHIVELOG mode and archive redo logs to multiple locations
3. Maintain multiple concurrent backups of the control file
4. Take frequent backups of physical datafiles and store them in a safe place, making multiple copies if possible

As long as users have backups of the database and archive redo logs in safe storage, the original database can be recreated.

4.3 LOGICAL DATABASE BACKUP

Oracle utility Import/Export are used to perform Logical Database Operation, which allow us to make exports & imports of the data objects, and transfer the data across databases that reside on different hardware platforms on different Oracle versions.Export (exp) and import (imp) utilities are used to perform logical database backup and recovery. When exporting, database objects are dumped to a binary file which can then be imported into another Oracle database.

From Oracle 10g, users can choose between using the old imp/exp utilities, or the newly introduced Data pump utilities, called expdp and impdp. These new utilities introduce much needed performance improvements, network based exports and imports, etc.

Various parameters are available to control what objects are exported or imported. To get a list of available parameters, run the exp or imp utilities with the help=yes parameter.

The export/import utilities are commonly used to perform the following tasks:

- Backup and recovery (small databases only)
- Move data between Oracle databases on different platforms.
- Reorganization of data/ eliminate database fragmentation (export, drop and re-import tables)
- Upgrade databases from extremely old versions of Oracle
- Detect database corruption. Ensure that all the data can be read
- Transporting tablespaces between databases

A. Different Modes of Export/Import Utility

1. **Full Export:** The EXP_FULL_DATABASE and IMP_FULL_DATABASE, respectively, are needed to perform a full export. Use the full export parameter for a full export.
2. **Tablespace:** Use the tablespaces export parameter for a tablespace export.
3. **User:** This mode can be used to export and import all objects that belong to a user. Use the owner export parameter and the fromuser import parameter for a user (owner) export-import.
4. **Table:** Specific tables (and partitions) can be exported/imported with table export mode. Use the tables export parameter for a table export.

4.3.1 EXPORT UTILITY

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

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

Exported dump files can be read only by using the Import utility of Oracle. We cannot use earlier versions of import utility for importing the data exported using current version.

EXP command can be used to invoke export utility interactively without any parameters. Parameters also can be specified in a file called parameter file. We can use more than one parameter file at a time with exp command.

General Parameters are used with exp **command are as:**

- **Full:** Use this parameter to specify full export mode.
- **Tablespaces:** Use this parameter to specify tablespace export mode.
- **Owner:** Use this parameter to specify user export mode.
- **Tables:** Use this parameter to specify table export mode.
- **Query:** Restricts the exported rows by means of a where clause. The query parameter can only be used for table export mode. For obvious reasons, it must be appliable to all exported tables.
- **Parfile:** Specifies a parfile. Parameter file is a simple text files creating using any text editor.

There are basically 3 types of exports like Full, Owner, and Table. **Full export** exports all the objects, structures and data within the database for all schemas. **Owner export** exports only the objects owned by specific user account. **Table export** exports only tables owned by a specific user account.

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

Examples:

1. We want to export EMP table from scott/tiger (username and password respectively) users and exported data will be stored into dump file namely emp as a command line parameter.

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

2. We want to export EMP table from scott/tiger (username and password respectively) users and exported data will be stored into dump file namely emp in interactive mode.

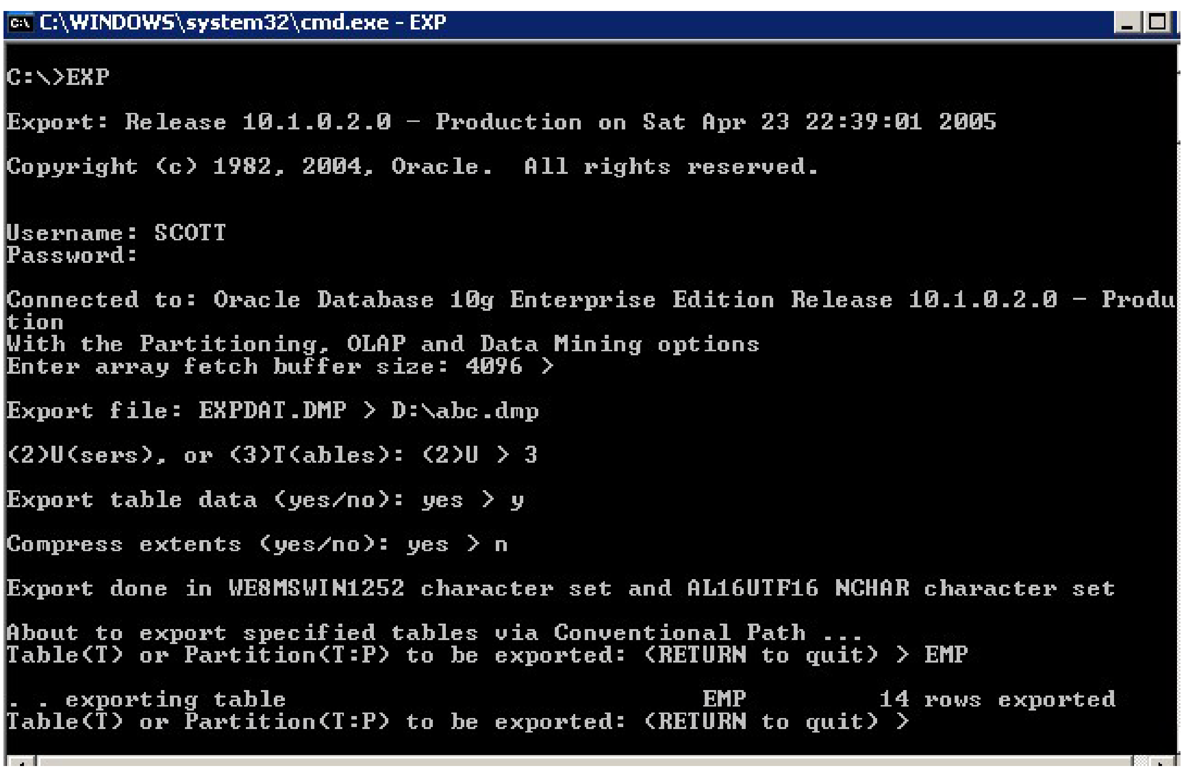


Figure 8.1: Exporting single table in interactively mode.

3. We want to export emp table with deptno=10 in non-interactive mode.

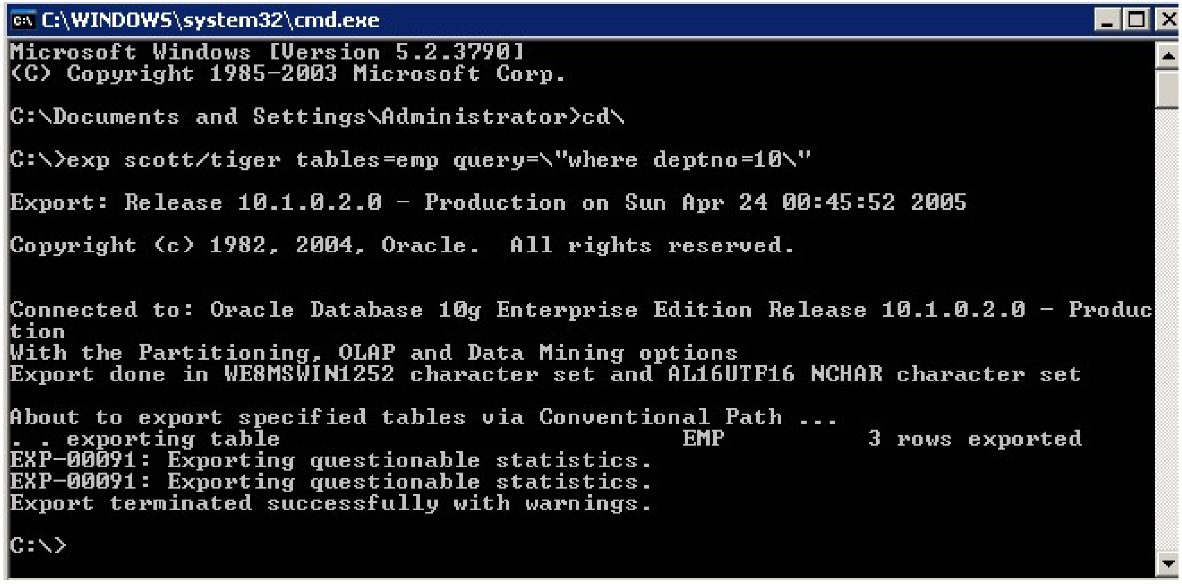


Figure 8.2: Exporting conditional rows in non-interactively mode.

4.3.2 IMPORT UTILITY

IMP command can be used to invoke import utility interactively without any parameters. Import utility is used to extract objects from export dump file created using export utility. We c an us e m ore t han o ne par ameter f ile at a time w ith exp c ommand. Various parameters of Import Utility are described as follow:

- FFER:**The integer specified for BUFFER is the size, in bytes, of the buffer through which data rows are transferred.**
- COMMIT:**Specifies whether Import should commit after each array insert. By default, Import commits only after loading each table, and Import performs a rollback when an error occurs, before continuing with the next object.**
- CONSTRAINTS: **Specifies whether or not table constraints are to be imported. The default is to import constraints. If you do not want constraints to be imported, you must set the parameter value to n.**
- FILE:**Specifies the names of the export files to import. The default extension is .dmp, because Export supports multiple export files, you may need to specify multiple filenames to be imported.**
- FROMUSER:**The parameter enables you to import a subset of schemas from an export file containing multiple schemas.**
- FULL: **Specifies whether to import the entire export dump file.**
- GRANTS:**Specifies whether to import object grants.**
- PARFILE:**Specifies a filename for a file that contains a list of Import parameters. For more information about using a parameter file, see Parameter Files.**
- ROWS:**Specifies whether or not to import the rows of table data.**
- TABLES:**Specifies that the import is a table-mode import and lists the table names and partition and sub partition names to import. Table-mode import lets you import entire partitioned or non-partitioned tables.**
- TOUSER: **Specifies a list of user names whose schemas will be targets for Import. The user names must exist prior to the import operation; otherwise an error is returned. The IMP_FULL_DATABASE role is required to use this parameter. To import to a different schema than the one that originally contained the object, specify TOUSER.**
- USERID: **Specifies the username/password (and optional connect string) of the user performing the import.**

```
C:\WINDOWS\system32\CMD.exe

C:\>IMP SCOTT/TIGER  FILE=D:\file1.dmp  TABLES= (TEST)

Import: Release 10.1.0.2.0 - Production on Wed Apr 27 23:06:24 2005

Copyright (c) 1982, 2004, Oracle.  All rights reserved.


Connected to: Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Produc
tion
With the Partitioning, OLAP and Data Mining options

Export file created by EXPORT:V10.01.00 via conventional path
import done in WE8MSWIN1252 character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into SCOTT
. . importing table                        "TEST"          3 rows imported
Import terminated successfully without warnings.

C:\>
```

Figure 8.3: Example of Import Utility in Interactive mode.

It is possible to import dump created using an earlier (version 8.1.7 utility) version can be imported using the later version utility (Version 9.0.1 utility). We should not use later version utilities to export data from earlier database versions. But an earlier utility can be used to export later versions of database. For example you can export data from Oracle9i using 8.1.7 utility and can import that exported file into oracle 8i database using import utility 8.1.7.

4.4 PHYSICAL DATABASE BACKUP

4.4.1 BACKUP

Backups can be combined in a variety of ways. For example, we can take weekly whole database backups, to ensure a relatively current copy of original database information, but take daily backups of the most accessed tablespaces. The DBA can also multiplex the all important control file and archived redo log as an additional safeguard.

A. **Online Database Backup:** An online backup or also known as an open backup is a backup in which all read-write datafiles and control files have not been check pointed with respect to the same SCN. If the database must be up and running 24 hours a day, 7 days a week, then you have no choice but to perform online backups of a whole database which is in ARCHIVELOG mode.

B. Offline Database Backup: In this backup, all datafiles and control files are consistent to the same point in time - consistent with respect to the same SCN. This type of backup allows the user to open the set of files created by the backup without applying redo logs, since the data is already consistent. The only way to perform this type of backup is to shut down the database cleanly and make the backup while the database is closed. A consistent whole database backup is the only valid backup option for databases running in NOARCHIVELOG mode.

Whole Database Backup: The most common type of backup, a whole database backup contains the control file along with all database files that belong to a database. If operating in ARCHIVELOG mode, the DBA also has the option of backing up different parts of the database over a period of time, thereby constructing a whole database backup piece by piece.

Tablespace Backups: A tablespace backup is a subset of the database. Tablespace backups are only valid if the database is operating in ARCHIVELOG mode. The only time a tablespace backup is valid for a database running in NOARCHIVELOG mode is when that tablespace is read-only or offline-normal.

Datafile Backups: A datafile backup is a backup of a single datafile. Datafile backups, which are not as common as tablespace backups and are only valid if the database is run in ARCHIVELOG mode. The only time a datafile backup is valid for a database running in NOARCHIVELOG mode is if that datafile is the only file in a tablespace.

Control File Backups: A control file backup is a backup of a database's control file. If a database is open, the user can create a valid backup by issuing the following SQL statement: **ALTER DATABASE BACKUP CONTROLFILE** to 'location'; or use Recovery Manager (RMAN).

Archived Redo Log Backups: Archived redo logs are the key to successful media recovery. Depending on the disk space available and the number of transactions executed on the database, you want to keep as many days of archive logs on disk and you want to back them up regularly to ensure a more complete recovery.

Configuration Files: Configuration files may consist of spfile or init.ora, password file, tnsnames.ora, and sqlnet.ora. Since these files do not change often, then they require a less frequent backup schedule. If you lost a configuration file it can be easily recreated manually. When restore time is a premium, it will be faster to restore a backup of the configuration file then manually creating a file with a specific format.

4.4.1.1 Types of Backup

There are basically two types of Backup we can take for Oracle Database.

I. **OFFLINE Backup**

When database is DOWN, no activity running on database, no one accessing the database, that time taken database backup called OFFLINE BACKUP. It is also known as **offline** or **consistent** database backup. Database doesn't require ARCHIVELOG mode for COLD backup. To take offline backup we must need to SHUTDOWN Oracle Database and stop Database service.

II. **ONLINE Backup**

When database is open, user accessing the database that time we taken backup is called online, hot or inconsistent backup. Database must require ARCHIVELOG mode for HOT backup.

Making User-Managed Backups of Online Tablespaces and Datafiles

You can back up all or only specific datafiles of an online tablespace while the database is open. The procedure differs depending on whether the online tablespace is read/write or read-only. You should not back up temporary tablespaces.

Making User-Managed Backups of Online Read/Write Tablespaces

You must put a read/write tablespace in backup mode to make user-managed datafile backups when the tablespace is online and the database is open. The **ALTERTABLESPACE ... BEGINBACKUP** statement places a tablespace in backup mode. In backup mode, the database copies whole changed data blocks into the redo stream. After you take the tablespace out of backup mode with the **ALTERTABLESPACE...ENDBACKUP** or **ALTERDATABASEENDBACKUP** statement, the database advances the datafile header to the current database checkpoint.

When restoring a datafile backed up in this way, the database asks for the appropriate set of redo log files to apply if recovery be needed. The redo logs contain all changes required to recover the datafiles and make them consistent.

To back up online read/write tablespaces in an open database:

1. Before beginning a backup of a tablespace, identify all of the datafiles in the tablespace with the DBA_DATA_FILES data dictionary view.
2. Mark the beginning of the online tablespace backup. For example, the following statement marks the start of an online backup for the tablespace users:
   ALTER TABLESPACE users BEGIN BACKUP;
3. Back up the online datafiles of the online tablespace with operating system commands.
4. After backing up the datafiles of the online tablespace, run the SQL statement ALTERTABLESPACE with the ENDBACKUP option.
   ALTER TABLESPACE users END BACKUP;

5. Archive the un-archived redo logs so that the redo required to recover the tablespace backup is archived.
   ALTER SYSTEM ARCHIVE LOG CURRENT;

Making User-Managed Backups of the Control File

Back up the control file of a database after making a structural modification to a database operating in ARCHIVELOG mode. To back up a database's control file, you must have the ALTERDATABASE system privilege.

Backing Up the Control File to a Binary File

The primary method for backing up the control file is to use a SQL statement to generate a binary file. A binary backup is preferable to a trace file backup because it contains additional information such as the archived log history, offline range for read-only and offline tablespaces, and backup sets and copies (if you use RMAN). Note that binary control file backups do not include tempfile entries.

To back up the control file after a structural change:

- Back up the database's control file, specifying a filename for the output binary file.

ALTER DATABASE BACKUP CONTROLFILE TO '/disk1/backup/cf.bak' REUSE;

Specify the REUSE option to make the new control file overwrite one that currently exists.

Making User-Managed Backups of Archived Redo Logs

To save disk space in your primary archiving location, you may want to back up archived logs to tape or to an alternative disk location. If you archive to multiple locations, then only back up one copy of each log sequence number.

4.4.2 RECOVERY

Basic recovery involves two parts: restoring a physical backup and then updating it with the changes made to the database since the last backup. The most important aspect of recovery is making sure all data files are consistent with respect to the same point in time. Oracle has integrity checks that prevent the user from opening the database until all data files are consistent with one another.

A. RECOVERY PROCESS

In every type of recovery, Oracle sequentially applies redo data to data blocks. Oracle uses information in the control file and datafile headers to ascertain whether recovery is necessary. Recovery has two parts: rolling forward and rolling back. When Oracle rolls forward, it applies redo records to the corresponding data blocks. Oracle systematically goes through the redo log to determine which changes it needs to apply to which blocks, and then changes the blocks. For example, if a user adds a row to a table, but the server crashes before it can save the change to disk, Oracle can use the redo record for this transaction to update the data block to reflect the new row.

Once Oracle has completed the rolling forward stage, the Oracle database can be opened. The rollback phase begins after the database is open. The rollback information is stored in transaction tables. Oracle searches through the table for uncommitted transactions, undoing any that it finds. For example, if the user never committed the SQL statement that added the row, then Oracle will discover this fact in a transaction table and undo the change.

- **Responding to the Loss of a Subset of the Current Control Files**

Use the following procedures to recover a database if a permanent media failure has damaged one or more control files of a database and at least one current control file has not been damaged by the media failure.

- Copying a Multiplexed Control File to a Default Location

If the disk and file system containing the lost control file are intact, then you can simply copy one of the intact control files to the location of the missing control file. In this case, you do not have to edit the CONTROL_FILES initialization parameter.

- **To replace a damaged control file by copying a multiplexed control file:**

If the instance is still running, then shut it down:

SQL> SHUTDOWN ABORT

Correct the hardware problem that caused the media failure. If you cannot repair the hardware problem quickly, then proceed with database recovery by restoring damaged control files to an alternative storage device.

Use an intact multiplexed copy of the database's current control file to copy over the damaged control files.

Start a new instance and mount and open the database.

SQL> STARTUP

- **Determining Which Datafiles Require Recovery**

You can use the dynamic performance view V$RECOVER_FILE to determine which files to restore in preparation for media recovery. This view lists all files that need to be recovered, and explains why they need to be recovered.

The following query displays the file ID numbers of datafiles that require media recovery as well as the reason for recovery (if known) and the SCN and time when recovery needs to begin:

SELECT * FROM V$RECOVER_FILE;

Query V$DATAFILE and V$TABLESPACE to obtain filenames and tablespace names for datafiles requiring recovery.

- **Restoring Datafiles**

If a media failure permanently damages one or more datafiles of a database, then you must restore backups of these datafiles before you can recover the damaged files. If you cannot restore a damaged datafile to its original location (for example, you must replace a disk, so you restore the files to an alternate disk), then you must indicate the new locations of these files to the control file.

If you are restoring an Oracle file on a raw disk or partition, then the procedure is basically the same as when restoring to a file on a file system. However, you must be aware of the naming conventions for files on raw devices (which differ depending on the operating system), and use an operating system utility that supports raw devices.

To restore backup datafiles to their default location:

1. Determine which datafiles to recover by using the techniques described in "Determining Which Datafiles Require Recovery".

2. If the database is open, then take the tablespaces containing the inaccessible datafiles offline.
   ALTER TABLESPACE users OFFLINE IMMEDIATE;

3. Copy backups of the damaged datafiles to their default location using operating system commands.

4. Recover the affected tablespace. For example, enter:
   RECOVER TABLESPACE users

5. Bring the recovered tablespace online. For example, enter:
   ALTER TABLESPACE users ONLINE;

Recovering After the Loss of Archived Redo Log Files:

If the database is operating in ARCHIVELOG mode, and if the only copy of an archived redo log file is damaged, then the damaged file does not affect the present operation of the database. The following situations can arise, however, depending on when the redo log was written and when you backed up the datafile.

- **Check Your Progress**

10. Describe Basic Principles for Backup Strategy?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

11. Which role has to grant for Full Database Export/Import?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

12. Which Parameter of Import Utility is used to Prevent rollback when error occurs ?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

13. What do you mean by Inconsistent Backup?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

14. How to find File names and Tablespace names for datafile requiring recovery?

…………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

4.5 LET US SUM UP

In this chapter, we have discussed about different types or Database Backup Strategies like Logical Backup and Physical Backup. In which conditions we have to perform logical backup. We have also learnt different parameters for Import/Export utility of Oracle. Also we have different types of physical backup like hot and cold backup and try to describe all the possible aspects of both types of physical backups and recovery strategies.

4.6 CHECK YOUR PROGRESS: POSSIBLEANSWERS

1. Basic principles follow these four simple steps:
   - Multiplex the online redo logs
   - Run the database in ARCHIVELOG mode and archive redo logs to multiple locations
   - Maintain multiple concurrent backups of the control file
   - Take frequent backups of physical datafiles and store them in a safe place, making multiple copies if possible
2. The EXP_FULL_DATABASE and IMP_FULL_DATABASE, respectively, are needed to perform a full export.
3. COMMIT specifies whether Import should commit after successfully execution of Import.
4. Inconsistent Backup means a backup taken when database is open and database must require ARCHIVELOG mode for it. It is also known as HOT Backup.
5. V$DATAFILE and V$TABLESPACE data dictionary is used to obtain filenames and tablespace names for datafiles requiring recovery

4.7 ASSIGNMENTS

1. Explain Different Command line Parameters for EXPORT with example.
2. Explain Different Command line Parameters for IMPORT with example.
3. Define Online Backup? How can we Backup Read/Write Tablespace?
4. Explain Recovery Process in detail.

4.8 Further Reading

1. Oracle Database 11g : Backup and Recovery User's Guide, Lance Ashdown, Oracle Press.

2. Oracle Database 10g The Complete Reference, Kevin Loney, Oracle Press.

Block-3

Oracle Server and SQL

Unit 1: Structured Query Language

1

Unit Structure

1.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this chapter is to make the students,

- To understand SQL and its Process Architecture
- To learn various types of SQL Statements
- To understand SQL Operators & Functions.
- To learn Joins and Sub Queries in SQL.
- To Understand Views, Index and Sequence.

Outcome:

At the end of this unit,

- Students will be completely aware with Architecture of SQL.
- Students will come to know the SQL statements in detail.
- Students will be able to write queries to retrieve data from tables as per organization requirements.
- Students will be able to create different SQL objects like Tables, Views, Indexes etc.

1.2 INTRODUCTION

SQL is an ANSI standard computer language, which is used for storing, manipulating and retrieving data stored in relational database. SQL is the standard language for Relational Database System.

SQL Process

When executing an SQL commands, system first determines the best way to carry out SQL query request and SQL engine figure out how to interpret the task. There are various components included in the process which is known as Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine etc.

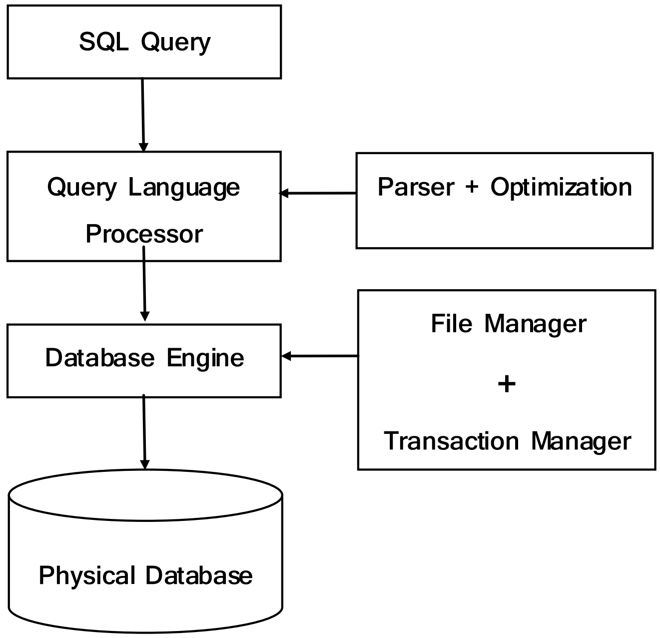


Figure-9.1 Simple diagram of SQL

Above figure shows that when SQL Query will fire first Query Language Process parses and optimize SQL query and pass the optimized version into the Database engine.

1.3 BASIC DATA TYPES OF SQL

Oracle Database provides following basic data types for attributes defined with CREATE TABLE clause of database.

Data Types	**Description**
Char (N)	Fixed Length Character Data. **Maximum size is 2000 bytes.** Default or Minimum Size 1 Byte.
Varchar (N)	Variable Length Character Data. **Maximum up to 2000 characters.**
Varchar2 (N)	Variable Length Character Data. **Maximum**

	up to 4000 characters.
Nvarchar2 (N)	Variable-length Unicode character string having maximum size is determined by the national character set definition, with an upper limit of **4000 bytes**.
Number (P,S)	Numeric data type for integers and Real Numbers. **P = Overall number of Digits. Maximum values 38. S = Number of digits to the right of the decimal point.**
FLOAT (p)	A subtype of the NUMBER data type. A FLOAT value requires from **1 to 22 bytes**.
LONG	Variable Length Character Data (**Up to 2GB**)
Date	Date data type for storing date and time. The size is fixed at **7 bytes**.
BINARY_FLOAT	32-bit floating point number.
BINARY_DOUBLE	64-bit floating point number.
RAW & LONG RAW	RAW Binary Data **RAW:** Maximum size is **2000 bytes**. **LONG RAW:** Maximum up to **2GB**
CLOB	Character Data (**Up to 4GB**)
NCLOB	Character Data containing Unicod characters. (**Up to 4GB**)
BLOB	Binary Data (**Up to 4GB**)
BFILE	Binary Data stored into external file (**Up to 4GB**)
ROWID	A **base-64 number system** representing the unique address of a raw in its table.
UROWID	A **base-64 number system** representing

	the logical address of a raw of an **indexed organized table**.
DATETIME Data Types	
TIMESTAMP	Date with Fractional Seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months.
INTERVAL DAY TO SECOND	Stored as an interval of days, hours, minutes and seconds.

1.4 SQL STATEMENTS

SQL statement includes data insert, query, update and delete, schema creation and modification and data access control. Based upon that SQL statements are divided into different categories as described below:

Data Manipulation Language (DML)	
SELECT	Retrieve certain record from one or more tables or views.
INSERT	Create new record into the table.
UPDATE	Modify existing record(s).
DELETE	Delete existing record(s).
MERGE	Conditionally insert or update data depending on its presence, also known as UPSERT.

Data Definition Language (DDL)	
CREATE	Create New Objects in Database like Table, View Index, etc.
ALTER	Modify the existing object.
DROP	Destroying an existing object.
RENAME	Change the name of existing object.
TRUNCATE	Deleting an existing object. (Drop and Re-Create)
COMMENT	Provides Single line or multi line comment(s).

Data Control Language (DCL)	
GRANT	Gives different Privileges to the user.
REVOKE	Tack back privileges which is previously granted from user.

Transaction Control Language (TCL)	
COMMITE	Make permanent all changes performed in the transaction.
ROLLBACK	Undo all uncommitted works done by the transaction(s).
SAVEPOINT	Identify a point in a transaction to which you can later roll back.

1.5 DATA DEFINITION STATEMENTS

Data Definition Statements of the SQL is used to create different database objects and manage that objects.

1.5.1. CREATE TABLE

Create Table clause is used to create a new database objects like table, view, index etc.
Syntax:

CREATE TABLE <TABLE NAME>
(
<Column 1><Data type><Size> [not null] [unique] [<column constraint>],
<Column 2><Data type><Size> [not null] [unique] [<column constraint>],
----
<Column N><Data type><Size> [not null] [unique] [<column constraint>],
[Table Constraint(s)]
);

For each column, a name and a data type must be specified and the column name must be unique within table definition. Columns are separated by colons.

1.5.2. ALTER TABLE

ALTER TABLEcommand is used to add, delete or modify columns in an existing table. You would also use ALTER TABLE command to add and drop various constraints on an existing table.

Syntax:

ALTER TABLE <TABLE NAME> ADD/MODIFY/DROP column [datatype];

1.5.3. DROP TABLE

It is used to delete remove entire table with structure from the database.

Syntax:

DROP TABLE <TABLE NAME> ;

1.5.4. TRUNCATE TABLE

The TRUNCATE TABLE command is used to delete complete data from an existing table.

Syntax:

TRUNCATE TABLE <TABLE NAME> ;

Example:

1. Create Salesman Table with Salesman No as a Primary Key and Salesman Name as a mandatory field.

```
CREATE TABLE SALESMAN
  (
        SNUM            NUMBER (4) PRIMARY KEY,
        SNAME           VARCHAR2(30) NOT NULL,
        CITY            VARCHAR2(30),
        COMM            NUMBER(4,2)
  );
```

2. Add New Column Mobile No into Salesman Table.
   ALTER TABLE SALESMAN ADD (MOBILE NUMBER (10));

3. Remove Customer Table.
   DROP TABLE CUSTOMER.

1.6 CONSTRAINTS

Constraints are the rules enforced on data columns on table. These are used to limit the types of data that can go into the table. Constraint could be applied at column level or table level. **Column level constraints** are applied only one column whereas **Table level constraints** are applied to the whole table. There are two types of data constraints that can be applied to data being inserted into the tables.

1.6.1. I/O CONSTRAINTS

This data constraint determines the speed at which data can be inserted or extracted from a table.

A. PRIMARY KEY

Primary key is a filed in a table which is uniquely identifies each row (or record) in a database table. Primary key field must be mandatory means can't have null values and must be unique values. A table can have only one primary key, which may consist of single or multiple fields. When Primary key created on single field it is known as **Single Field Primary Key** and when Primary key created on multiple fields it is known as **Composite Primary Key.**

Examples:

1. Single Field Primary Key at Column Level:

Below example shows the Salesman table with SNUM as Primary key created at column level.

```
CREATE TABLE SALESMAN
(
    SNUM        NUMBER (4) PRIMARY KEY,
    SNAME       VARCHAR2(30) NOT NULL,
    CITY        VARCHAR2(30),
    COMM             NUMBER(4,2)
);
```

2. Composite Primary Key at Table Level:

Below example shows the Salesman table with SNUM and BCODE as Composite Primary key.

```
CREATE TABLE SALESMAN
(
    SNUM        NUMBER (4),
    BCODE       NUMBER (4),
    SNAME       VARCHAR2(30) NOT NULL,
    CITY        VARCHAR2(30),
    COMM             NUMBER(4,2),
    PRIMARY KEY (SNUM,BCODE)
);
```

B. FOREIGN KEY / REFERENCE KEY

Foreign key (or reference key) is a column or a combination of columns whose values match a Primary key in a different table. The relationship between tables matches the primary key in one of the tables with foreign key in other tables. The referencing table is called the child table, and the referenced table is called the parent table.

Examples:

1. Reference Key at Column Level:

```
CREATE TABLE CUSTOMER
(
 CNUM      NUMBER (4) PRIMARY KEY,
 CNAME     VARCHAR2(30) NOT NULL,
 CITY      VARCHAR2(30),
 RATTING   NUMBER(3),
 SNUM      NUMBER (4) CONSTRAINT FK_SNUM REFERENCES SALESMAN
);
```

In this example, the column SNUM of CUSTOMER table (Child Table) builds the foreign key namely FK_SNUM and references the Primary key of SALESMAN table (Parent Table).

2. Reference Key at Table Level:

```
CREATE TABLE CUSTOMER
(
 CNUM      NUMBER (4) PRIMARY KEY,
 CNAME     VARCHAR2(30) NOT NULL,
 CITY      VARCHAR2(30),
 RATTING   NUMBER(3),
 SNUM      NUMBER (4),
 CONSTRAINT FK_SNUM FOREIGN KEY (SNUM) REFERENCES SALESMAN
(SNUM)
);
```

1.6.2. BUSINESS RULE CONSTRAINTS

Business Rule constraints allow application of business rules to table columns. These rules are applied to data, prior the data is being inserted into the table columns.

A. UNIQUE

The UNIQUE constraint prevents duplicate values in the column. But it permits multiple NULL values in the column. Same as primary key unique constraint also create unique index on the field.

Examples:

Unique Key at Column Level:

```
CREATE TABLE CUSTOMER
(
 CNUM      NUMBER (4) PRIMARY KEY,
 CNAME     VARCHAR2(30) NOT NULL,
 CITY      VARCHAR2(30),
 EMAIL     VARCHAR2(30) CONSTRAINT CUST_EMAIL_UK UNIQUE,
 RATTING   NUMBER(3),
 SNUM      NUMBER (4) CONSTRAINT FK_SNUM REFERENCES
SALESMAN
);
```

B. NOT NULL

In oracle, by default column can hold NULL values. If you do not want a column to have a NULL values, then you need to define NOT NULL constraint on that column. NOT NULL constraints only implemented at column level.

Examples:

```
CREATE TABLE CUSTOMER
(
CNUM      NUMBER (4) PRIMARY KEY,
CNAME     VARCHAR2(30) NOT NULL,
CITY      VARCHAR2(30),
EMAIL          VARCHAR2(30) CONSTRAINT CUST_EMAIL_UK
UNIQUE,
RATTING   NUMBER(3) NOT NULL,
```

```
 SNUM         NUMBER (4 ) CONSTRAINT FK_SNUM REFERENCES
 SALESMAN
 );
```

C. CHECK CONSTRAINT

Business Rule validations can be applied to a table column by using check constraint. Check constraint must be specified as a logical expression that evaluates either to TRUE or FALSE.

Examples:

Check constraint at Table Level:

```
CREATE TABLE CUSTOMER
(
  CNUM       NUMBER (4) PRIMARY KEY,
  CNAME      VARCHAR2(30) NOT NULL,
  CITY       VARCHAR2(30),
  RATTING    NUMBER(3),
  SNUM       NUMBER (4) CONSTRAINT FK_SNUM REFERENCES
  SALESMAN,
  CONSTRAINT CUST_NAME_CHK CHECK (CNAME = UPPER (CNAME)),
  CONSTRAINT CUST_RATTING_CHK CHECK (RATING >= 100)
);
```

Above example create CUSTOMER table, where Name of customer must be consist of upper case letters only and minimum ratting of customer is 100.

D. DEFAULT VALUE

The DEFAULT constraint provides a default value to a column when a record is loaded into the table, and the column is left empty.

Examples:

```
CREATE TABLE CUSTOMER
(
    CNUM        NUMBER (4) PRIMARY KEY,
    CNAME       VARCHAR2(30) NOT NULL,
    CITY        VARCHAR2(30),
    RATTING     NUMBER(3) DEFAULT 100,
    SNUM        NUMBER (4) CONSTRAINT FK_SNUM REFERENCES
    SALESMAN
);
```

Above example create CUSTOMER table with RATTING field is set to 100 by default.

1.7 DATA MANIPULATION STATEMENTS

1.7.1. INSERT INTO STATEMENT

Insert Into statement is used to insert records into the database table. The General syntax of INSERT INTO clause as given below:

INSERT INTO <TABLE NAME> [(Column1, Column2 ..., ColumnN)]
VALUES (Value1, Value2..., ValueN)

Here, column1, column2 ..., columnN are the names of the columns in the table into which you want to insert data. You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table.

Example:

1. INSERT INTO SALESMAN VALUES (1001, 'BADAL', 'PATAN', 0.12);
2. INSERT INTO SALESMAN (SNUM, SNAME, COMM) VALUES (1002, 'VIRAL', 0.09);

1.7.2. UPDATE STATEMENT

The **UPDATE** Query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update selected rows, otherwise all the rows would be affected. General Syntax of Update Clause as:

UPDATE <TABLE_NAME> SET column1 = value1, column2 = value2....
WHERE [condition];

Example:

1. UPDATE SALESMAN SET CITY = 'PATAN' WHERE SNUM = 1002;

1.7.3. DELETE STATEMENT

The **DELETE** Query is used to delete the existing records from a table. Syntax of Delete Statement as given below:

DELETE FROM <TABLE_NAME> WHERE [condition];

Example:

1. DELETE FROM SALESMAN WHERE SNUM = 1002;

1.7.4. SELECT STATEMENT

SQL SELECT Statement is used to fetch record(s) from existing database table(s), which returns the result data in form of table. When we will display selected columns from the table then it is known as **Projection operations**.

Syntax:

SELECT [DISTINCT] column1, column2 ... FROM <FROM_CLAUSE>

[WHERE <CONDITION>]

[GROUP BY <EXPRESSION >]

[HAVING <CONDITION>]

[ORDER BY <COLUMN> [ASC|DESC]]

Example:

1. Display all the information of salesman's in the sequence of City, Name and comm.
   SELECT CITY, SNAME, COMM FROM SALESMAN;

1.7.5. WHERE CALUSE IN SQL

WHERE clause in query represents the condition for fetching records from the table(s), known as **SELECTION** operation.

Example:

1. Display Num and Name of all customers with salesman number 1001.
   SELECT CNUM, CNAME, SNUM FROM CUSTOMER WHERE SNUM = 1001;

1.7.6. ORDER BY CLAUSE

The SQL Order By Clause is used in SELECT statement to sort the data either in ascending or descending order, based on one or more columns. Oracle sorts query results in ascending order by default. If you want to sort the data in descending order, you must explicitly specify using DESC Keyword follow the column name.

Example

1. List all Salesmen with commission above 10% and result should be in ascending order of City and reverse order of commission.

   SELECT SNUM,SNAME,CITY,COMM FROM SALESMAN WHERE COMM > 0.10 ORDER BY CITY, COMM DESC;

1.7.7. GROUP BY CLAUSE

The **SQL GROUP BY** clause establishes data groups based on columns and aggregates the information within a group only. The grouping criterion is defined by the columns specified in GROUP BY clause. GROUP BY clause can only be used with aggregate functions. The group by clause should contain all the columns in the select list expect those used along with the group functions.

Example

1. Display total orders for each salesman.
   SELECT SNUM, SUM (AMOUNT) FROM ORDERS GROUP BY SNUM;

1.7.8. HAVING CLAUSE

The Having Clause enables you to specify conditions that filter which group results appear in the final results. HAVING clause places conditions on groups created by the GROUP BY clause. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Example

1. Display total orders of each salesman having more than single order.
   SELECT SNUM, COUNT (ONUM) FROM ORDERS GROUP BY SNUM HAVING COUNT(SNUM) > 1;

1.8 SQL OPERATORS

An operator is a reserved word used primarily in SQL Statement's to perform operation(s). An operator manipulates individual data items and returns a result. The data items are called **operands or arguments**.

A. **Arithmetic Operator:** Arithmetic operators manipulate numeric operands. Below Tables shows the list of Arithmetic Operators.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

B. Character Operator: Character operators are used in expressions to manipulate character strings. Below Tables shows the list of Character Operators.

Operator	Description
||	Concatenates character strings

C. Comparison Operator: Comparison operators are used in conditions that compare one value or expression with another. The result of a comparison can be TRUE or FALSE.

Operator	Description
=	Equality test.
!=, ^=, <>	Inequality test.
>	Greater than test.
<	Less than test.
>=	Greater than or equal to test.
<=	Less than or equal to test.
IN	"Equivalent to any member of" test. Equivalent to "= ANY".
ANY/ SOME	Compares a value to each value in a list or returned by a query. Evaluates to FALSE if the query returns no rows.
NOT IN	Equivalent to "!= ANY". Evaluates to FALSE if any member of the set is NULL.
ALL	Compares a value with every value in a list or

	returned by a query. Must be preceded by =, !=, >, <, < =, or > =. Evaluates to TRUE if the query returns no rows.
EXISTS	TRUE if a sub-query returns at least one row.
IS [NOT] NULL	Tests for nulls. This is the only operator that should be used to test for nulls.

D. Range Searching Operator: In order to select data that is within a range of values, the range searching operator is used.

Operator	**Description**
[Not] BETWEEN x AND y	[Not] greater than or equal to *x* and less than or equal to *y*.

E. Pattern Matching Operator: Pattern matching operator allows comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters.

Operator	**Description**
LIKE X	The character "%" matches any string of zero or more characters except null. The character " _" matches any single character.

F. Logical Operator: Logical operators manipulate the results of conditions.

Operator	**Description**
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE.
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE.
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE.

G. Set Operator: Set operators combine the results of two queries into a single result.

Operator	Description
UNION	Returns all distinct rows selected by either query.
UNION ALL	Returns all rows selected by either query, including all duplicates.
INTERSECT	Returns all distinct rows selected by both queries.
MINUS	Returns all distinct rows selected by the first query but not the second.

Example

1. **Display all customers not located in LONDON.**

   SELECT * FROM CUSTOMER WHERE CITY <> 'LONDON';

2. **List all salesmen with commission between 11% and 15%.**

   SELECT * FROM SALESMAN WHERE COMM BETWEEN 0.11 AND 0.15;

3. **List all salesmen whose names begin with letter 'B'.**

   SELECT * FROM SALESMAN WHERE SNAME LIKE 'B%';

1.9 ORACLE SQL BUILT-IN FUNCTIONS

Oracle SQL Built-in Functions serve the purpose of manipulating data items and returning a result. We can assign a value in form of variable or constants, such values are known as Arguments of functions. Oracle Functions can be divided into main two categories as described below:

1.9.1. GROUP FUNCTIONS (AGGREGATE FUNCTIONS)

These functions group the rows of data based on the values returned by the query. The group functions are used to calculate aggregate values, which return just one value after processing a group of rows.

Function	Value Returned
SUM (Values|Column)	Returns Sum of given Values.
AVG (Values|Column)	Return the Average Value.
COUNT (Values|Column)	Return Number of rows where the value of the column is not NULL
COUNT (*)	Return Number of rows including duplicates and NULLs
MAX (Values|Column)	Returns Maximum Value.
MIN (Values|Column)	Returns Minimum Value.
MEDIAN (Values|Column)	Returns Median (Middle) value in the sorted column, interpolating if necessary
STDDEV (Values|Column)	Returns Standard deviation of the column ignoring NULL values
VARIANCE (Values|Column)	Returns Variance of the column ignoring NULL values
CORR (Column-1,Column-2)	Returns Correlation coefficient between the two columns after eliminating nulls.

Example

1. Count the no. of salesmen currently having orders.

   SELECT COUNT(DISTINCT (SNUM)) FROM ORDERS;

1.9.2. SINGLE ROW FUNCTIONS (SCALAR FUNCTION)

Single row or Scalar functions return a value for every row that is processed in a query. There are four types of single row functions.

A. Numeric Functions: These are functions that accept numeric input and return numeric values.

Function	Value Returned
ABS (m)	Absolute value of m
MOD (m , n)	Remainder of m divided by n
POWER (m, n)	m raised to the nth power
ROUND (m , n)	m rounded to the nth decimal place
TRUNC (m, n)	m truncated to the nth decimal place
CEIL (n)	smallest integer greater than or equal to n
FLOOR (n)	greatest integer smaller than or equal to n
SQRT (n)	positive square root of n
EXP (n)	e raised to the power n
LN (n)	natural logarithm of n
LOG (n2, n1)	logarithm of n1, base n2
SIN (n)	sine (n)
COS (n)	cosine (n)
TAN (n)	tan (n)

B. String Functions: These are functions that accept character input and can return both character and number values.

Function	Value Returned
LOWER (s)	All letters are changed to lowercase.
UPPER (s)	All letters are changed to uppercase.
INITCAP (s)	First letter of each word is changed to uppercase and all other letters are in lower case.

CONCAT (s1, s2)	Concatenation of s1 and s2. Equivalent to *s1 || s2*
LPAD (s1, n , s2)	Returns s1 right justified and padded left with n characters from s2; s2 defaults to space.
RPAD (s1, n, s2)	Returns s1 left justified and padded right with n characters from s2; s2 defaults to space.
LTRIM (s,set)	Returns s with characters removed up to the first character not in set; defaults to space
RTRIM (s, set)	Returns s with final characters removed after the last character not in set; defaults to space
REPLACE (s, search_s, replace_s)	Returns s with every occurrence of search_s in s replaced by replace_s; default removes search_s
SUBSTR (s, m, n)	Returns a substring from s, beginning in position m and n characters long; default returns to end of s.
LENGTH (s)	Returns the number of characters in s.
INSTR (s1, s2, m, n)	Returns the position of the nth occurrence of s2 in s1, beginning at position m, both m and n default to 1.

C. Date Functions: These are functions that take values that are of datatype DATE as input and return values of datatype DATE.

Function	**Value Returned**
SYSDATE	Current date
LAST_DAY (Date)	Date of the last day of the month containing date
NEXT_DAY (Date, day)	Date of the first day of the week after date
ADD_MONTHS (Date, No. of Month)	Add No. of Months in Date

MONTHS_BETWEEN (Date-1, Date-2)	Returns Difference in Month between two dates.
GREATEST (Date-1, Date-2, ..., Date-N)	Latest of the given dates
LEAST (Date-1, Date-2, ..., Date-N)	Earliest of the given dates
NEW_TIME (Date,Current_Timezone,New_TimeZone)	Display Date and Time in New TimeZone Format

D. Conversion Functions: These are functions that help us to convert a value in one form to another form.

Function	Value Returned
TO_NUMBER (String, Format)	Character String converted to a Number as Specified by Format.
TO_CHAR(Value, Format)	Convert Number or Date into Character string as specified by Format.
TO_DATE (String, Format)	String Value converted in a Date as specified by given Format.
ROUND (Date, Format)	Date Rounded as specified by the Format.
TRUNC (Date, Format)	Date truncated as Specified by the Format.

1.10 SQL Joins

Sometimes it is required to retrieve information from multiple tables; at that time Join condition is required. Rows in one table can be joined to rows in another table according to common values existing in corresponding columns. We must have to keep in mind some principle as follows:

- When Writing a SELECT statement that joins tables, precede the column name with the table name for clarify and to enhance the database access.

- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join *N* tables together, you need a minimum of N-1 join conditions.

1.10.1. TYPES OF ORACLE JOINS

- **Inner Join**
- **Outer Join**
- **Self Join**

A. INNER Join (Equi Join OR Simple Join)

It is a simple SQL join condition which uses the equal sign as the comparison operator. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.

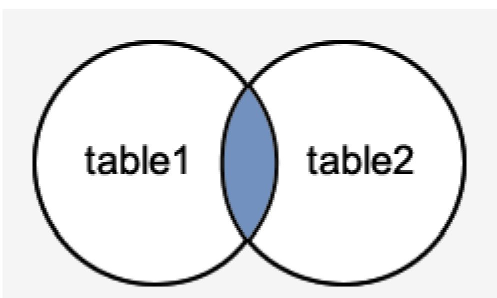


Figure-9.2 Inner Join Diagram

The SQL INNER JOIN would return the records where table1 and table2 intersect.

B. Outer Join

An Outer Join is used to identify situations where rows in one table do not match rows in a second table, even though the two tables are related. The SQL outer join operator in Oracle is (+) and is used on one side of the join condition only.

There are two types of outer joins:

- **LEFT OUTER JOIN**

- **RIGHT OUTER JOIN**

I. LEFT OUTER JOIN

A LEFT OUTER JOIN adds back all the rows that are dropped from the first (left) table in the join condition, and output columns from the second (right) table are set to NULL.

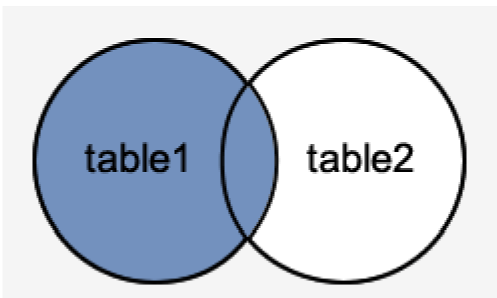


Figure-9.3 Left Outer Join Diagram

The SQL LEFT OUTER JOIN would return the all records from *table1* and only those records from *table2* that intersect with *table1*.

II. RIGHT OUTER JOIN

A RIGHT OUTER JOIN adds back all the rows that are dropped from the second (right) table in the join condition, and output columns from the first (left) table are set to NULL.

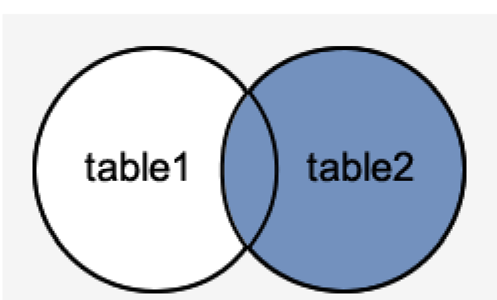


Figure-9.4 Right Outer Join Diagram

The SQL RIGHT OUTER JOIN would return the all records from *table2* and only those records from *table1* that intersect with *table2*.

C. Self Join

Sometimes you need to join a table to itself only. When a table is joined to itself, the join is known as Self Join. It is necessary to ensure that the join statement defines as alias for both copies of the table to avoid column ambiguity.

Example

1. Show the name of all customers with their relational salesman's name.

SELECT CUST.CNAME, SMAN.SNAME FROM CUSTOMER CUST, SALESMAN SMAN WHERE SMAN.SNUM = CUST.SNUM;

2. Find all pairs of customers having the same city without duplication.

SELECT CU.CNAME, CU.CITY, CUST.CNAME, CUST.CITY FROM CUSTOMER CU, CUSTOMER CUST WHERE CU.CITY = CUST.CITY AND CU.CNUM > CUST.CNUM;

1.11 SUB QUERIES

A query within another query is known as Sub query or Inner Query or Nested query. It is embedded within the WHERE clause. Sub queries must be enclosed within parentheses. A sub query is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Sub queries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators. There are a few rules that sub queries must follow:

- A sub query can have only one column in the SELECT clause, unless multiple columns are in the main query for the sub query to compare its selected columns.

- An ORDER BY cannot be used in a sub query, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a sub query.
- Sub queries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the sub query.

Example

1. Following example updates SALARY by 0.25 times in CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

```
UPDATE CUSTOMERS SET SALARY = SALARY * 0.25 WHERE AGE IN (SELECT
AGE FROM CUSTOMERS_BKP WHERE AGE >= 27 );
```

1.12 SQL VIEWS

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query. A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view.

CREATE VIEW <VIEW NAME> AS SELECT COLUMN1, COLUMN2..... FROM <TABLE NAME> WHERE [CONDITION];

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

DROP VIEW <VIEW NAME>;

1.13 SQL INDEXES

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

Syntax:

CREATE INDEX <INDEX_NAME> ON <TABLE_NAME>;

There are three types of index as follows:

- **Single-Column Indexes:** A single-column index is one that is created based on only one table column.

- **Unique Indexes:** Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table.
- **Composite Indexes:** A composite index is an index on two or more columns of a table.

An index can be dropped using SQL **DROP** command. Care should be taken when dropping an index because performance may be slowed or improved.

Syntax:

DROP INDEX <INDEX_NAME>;

1.14 SQL SEQUENCE

Sequence is an oracle object which is used to generate unique integers, which can help to generate primary keys automatically. A new primary key value can be obtained by selecting the most produced value and incrementing it. It required a lock during the transaction and causes other users to wait for next value of primary key it is known as **serialization**. To create a sequence users must obtain CREATE SEQUENCE system privileges.

Syntax:

```
CREATE SEQUENCE <SEQUENCE_NAME>
STARTWITH INITIAL-VALUE
INCREMENT BY INCREMENT-VALUE
MAXVALUE MAXIMUM-VALUE
CYCLE |NOCYCLE
CACHE | NOCACHE;
```

Where,

START WITH: Specifies the starting value for the Sequence.

INCREMENT BY: Specifies the value by which sequence will be incremented.

MAXVALUE: specifies the upper limit or the maximum value up to which sequence will increment itself.

CYCLE: Specifies that if the maximum value exceeds the set limit, sequence will restart its cycle from the beginning.

CACHE: Pre-allocates a set of sequence number and keep them into memory so the sequence number can be accessed faster.

Example

1. Let's start by creating a sequence, which will start from 1001, increment by 1 with a maximum value of 9999.

   ```
   CREATE SEQUENCE ST_SEQ
   STARTWITH1001
   INCREMENT BY1
   MAXVALUE 9999
   CYCLE;
   ```

   To insert Sequence Value in SNUM of Salesman table, query will be

   ```
   INSERTINTO SALESMAN VALUE (ST_SEQ.nextval,‘AMIT’, ’PATAN’, 0.15);
   ```

➢ **Check Your Progress**

15. Explain difference between varchar2 & nvarchar2 data types.

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

16. Explain difference between TRUNCATE and DROP Table.

……………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

17. What is Primary Key? Describe composite Primary Key with Example.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

18. What is Operator in SQL? List the different operators used in SQL.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

19. Define Aggregate and Scalar Function?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

20. What is Views in SQL?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

1.15 LET US SUM UP

In this chapter, we have discussed about SQL Architecture and different SQL Statements. We have also explored data types available in SQL. We have come to know vital processes like Selection, Projection Grouping, Joins and Sub Queries. We have also described different operators and functions available in SQL. We have tried to explore different constraints. We have described some SQL Objects like View, Indexes, and Sequences etc.

1.16CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Varchar2 represents variable length character data up to 4000 characters. While nvarchar2 represents Unicode character string having maximum size determined by the National Character Set with an upper limit of 4000 Bytes.
2. TRUNCATE clause is used to delete all records from existing tables. Definition of table remains as it is. While DROP removes entire definition of table means delete all records including the table structure.
3. Primary Key is used to uniquely identify each record in a database table. When Primary key is created on multiple fields of the table than it is known as Composite Primary Key. Composite Primary Key created at table level.

   Example:

   ```
   CREATE TABLE Employee
   (
           EmployeeId          NUMBER (4),
           BranchCode          NUMBER (4),
           EmployeeNAME        VARCHAR2(30) NOT NULL,
           EmployeeCity        VARCHAR2(30),
           EmployeeJoinDate    DATE,
           PRIMARY KEY (EmployeeId,BranchCode)
   );
   ```

   Above Query is used to Create Employee Table with Composite Primary Key namely (EmployeeId,BranchCode).
4. An operator is used to perform different operation and return result set. In SQL operators have different types as follows:
   A. Arithmetic Operators
   B. Character Operators
   C. Comparison Operators
   D. Range Searching Operator
   E. Pattern Matching Operator
   F. Logical Operator
   G. Set Operator

At the end of this unit,

• Students will be able to write simple procedure and execute it

• Students will write stored procedure for various operations to be applied on database table

• Students will be able to simple function and call it

2.2 INTRODUCTION

A procedure or function is a named object of PL/SQL block. There are two types of subprograms in PL/SQL namely Procedures and Functions. Every subprogram will contain declaration block, an execution block or body, and an exception handling block being an optional part.

When user executes a procedure or function, the execution takes place at the server side. This obviously reduces network traffic. The subprograms are the compiled programs and stored in the oracle database and can be invoked whenever required. Whenever the sub programs are called, they only need to execute because they are stored in compiled form. So, they save time required for compilation of the sub program.

2.3 STORED PROCEDURE BASICS

A procedure may take one or more arguments. If a procedure takes arguments then these arguments are to be supplied at the time of calling the procedure. A procedure contains two parts specification and the body. Procedure specification begins with Create and ends with procedure name or parameters list. Procedures without parameters are written without a parenthesis. The body of the procedure starts after the keyword IS or AS and ends with keyword End.

Syntax:

```
CREATE [OR REPLACE] PROCEDURE [schema.] procedure_name
[( parameter_1 [IN] [OUT] parameter_data_type_1,
parameter_2 [IN] [OUT] parameter_data_type_2,…
parameter_N [IN] [OUT] parameter_data_type_N )]
[AUTHID DEFINER | CURRENT_USER]
```

```
IS
— declaration_statements
BEGIN
— executable_statements
return {return_data_type};
[EXCEPTION
— the exception-handling statements]
END [procedure_name];
```

Where,

Create or Replace means the procedure is created if the procedure with the same name doesn't exist or the existing procedure is replaced with the new code.

IS represents the beginning of the body of the procedure and is similar to Declare in anonymous PL/SQL Blocks. The code between IS and BEGIN makes the Declaration section.

The syntax within the brackets [ ] indicate optional fields. The optional parameter list will contain name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.

EXCEPTION is again an optional part. It is used to handle run-time errors.

2.3.1 COMPONENTS OF PROCEDURE

To understand procedure easily we will divide the Procedure in two parts:

I. Procedure Head

All the code before the "IS" keyword is called the Procedure head or signature. Various parts of PL/SQL Procedure Head are:

A. Schema

This is an optional parameter and defines the schema name in which the procedure will be created. The default schema is the current user. If we specify a different user then, the other user must have the privileges to create a procedure in his/her schema.

B. Name

The NAME parameter defines the name of the procedure. The name of a procedure should be more meaningful and readable.

C. Parameters

The parameters are optional. These will be required to pass and receive values from a PLSQL procedure. There are 3 styles of passing parameters.

- **IN:** This is the default style of parameter in PLSQL procedure. We use the IN mode whenever we want the parameter to be read only i.e. we cannot change the value of the parameter in the PLSQL procedure.
- **OUT:** The OUT parameter returns the values to the calling subprogram orsubroutines. A default value cannot be assigned to OUT parameter so we cannot make it optional. We have to assign a value to OUT parameter before we exit the procedure or the value of the OUT parameter will be NULL. While calling a procedure with OUT parameters, we have to make sure that we pass variables for the corresponding OUT parameters.
- **IN OUT:** In this mode the actual parameter is passed to the PLSQL procedure with initial values and then within the PLSQL procedure the value of the parameter may get changed or reassigned. The IN OUT parameter is finally returned to the calling subroutine.

D. Authid

This is also an optional parameter and it defines whether the procedure will execute with the privileges of the Creator / Definer of the procedure or with that of the Current_User privileges.

II. Procedure's Body

Everything after the "IS" keyword is called the body of the procedure. The procedure's body contains the declaration of variables in the declaration section, the code to be executed in the executable statements part and the code to handle any exception in the exception handling part.

The declaration and exception handling parts are optional in PLSQL procedure body. We must have at least one executable statement in the executable statement part. The execution part is the one where we have to write the business logic. The Return statement in procedure is used to discontinue the execution of the procedure further and return the control to the calling subroutine.

To create a stored procedure, user must have Create Procedure system privilege. User must also have required object privileges on the objects that are referred in the procedure in order to successfully compile the procedure.

2.3.2 TYPES OF PARAMETERS

There are two types of parameters of a procedure.

1. Formal parameters
2. Actual Parameters

- **Formal Parameters**

The parameters declared in the definition of procedure are known as formal parameters. They receive the values sent while calling the procedure. For example,

- procedure Welcome (message varchar2, name varchar2)

In the above code message, name parameters are called as formal parameters.

- **Actual Parameters**

The values given within parentheses while calling the procedure are called as actual parameters.

- Welcome ('Welcome Mr.', 'Himanshu');

'Welcome Mr.' and ' Himanshu' are actual parameters. These values are copied to the corresponding formal parameters message and name.

2.4 CREATING STORED PROCEDURES

After discussing the different part of the procedure, its time to create procedure. Suppose we have a table named 'employee' as shown below:

```
Create table employee
        (Employee_id number(5),
        Employee_name varchar2(10),
        Employee_salary number(6,2),
```

```
                Employee_department varchar2(10),
                Employee_commission number(8,2));
```

After creating 'employee' table insert few records in it.

Now, we will create a Procedure in which we will pass the 'employee_id' and 'salary'. The Procedure will update the record of the employee having the same 'employee_id' using Oracle SQL Update statement.

Example:

```
Create or Replace Procedure update_employee_salary (emp_id_in IN Number,
salary_in IN Number)
IS
Begin
        Update employee
        Set employee_salary = salary_in
        Where employee_id = emp_id_in;
      dbms_output.put_line('Procedure executed successfully');
End update_employee_salary;
/
```

In the above code, we have created a procedure named 'update_employee_salary' which will take two parameters 'employee_id' and 'salary' and update the 'employee' table.

- **Calling PL/SQL Procedure**

After creating procedure, it can be called using the EXEC or EXECUTE Statement. Syntax to call a Procedure using EXEC or EXECUTE statement is:

Syntax:

```
EXEC procedure_name(parameters);
                         or
EXECUTE procedure_name(parameters);
```

Suppose, we want to update the salary of ‘employee_id = 101’ from 1000 to 1500 using update_employee_salary procedure. So, call update_employee_salary procedure using EXEC statement as shown below.

- Exec update_employee_salary(101,1500);

The procedure will successfully update the salary of employee having id ‘101’ from 1000 to 1500 using PL/SQL Procedure.

- **IN Parameter**

Here we will create a stored procedure to accept a single parameter and print out the message with parameter passed via DBMS_OUTPUT.

Example:

```
        Create or Replace Procedure INParameter(var in varchar2)
        IS
        Begin
         dbms_output.put_line(‘Welcome: The argument passed is: ‘ || var);
        End;
/
```

To Run the procedure pass following command with argument as stated in below:

- Exec INParameter(‘BAOU’);

Output:

- Welcome: The argument passed is: BAOU

- **OUT Parameter**

A stored procedure to demonstrate the OUT Parameter.

Example:

```
Create or Replace Procedure OUTParameter(outvar out varchar2)
IS
Begin
        outvar:= ‘Welcome to Hindustan’;
End;
/
```

Now execute the above procedure. It will create the procedure.

Now to execute the procedure we will write a following block of code and call the Procedure from the body of the block.

Example:

```
Declare
        outvar varchar2(100);
Begin
        outparameter(outvar);
        dbms_output.put_line(outvar);
End;
/
```

The executed code is shown below.

Output:

- Welcome to Hindustan

- **INOUT Parameter**

A stored procedure to accept a INOUT parameter (Param), construct the output message and assign back to the same parameter name(Param) again.

Example:

```
Create or replace procedure inoutparameter(param IN OUT varchar2)
IS
Begin
    param := 'Welcome to India ' || param;
End;
/
```

The executed code will create the procedure.

To execute the procedure we will create a following block of code and call the Procedure from the body of the block.

Example:

```
        Declare
                param varchar2(100) := 'veddesai';
        Begin
                inoutparameter(param);
                dbms_output.put_line(param);
        End;
/
```

The above code produces following output.

Output:

- Welcome to India veddesai

2.4.1 STORED PROCEDURE WITH DML STATEMENTS

I. INSERT Statement

First of all we will create User_data table in Oracle database as shown below.

```
Create Table User_data(
        User_id     number (5)   not null,   username     varchar2 (20)  not null,
        created_by   varchar2 (20)  not null,   created_date  date        not null,
  primary key (user_id) );
```

Once the table is created, we will create a stored procedure. The procedure will accept 4 IN parameters and insert it into table "User_data".

Example:

```
Create OR Replace Procedure insertUSERDATA(
                userid IN USER_data.USER_ID%TYPE,
                username IN USER_data.USERNAME%TYPE,
                createdby IN USER_data.CREATED_BY%TYPE,
                pdate IN USER_data.CREATED_DATE%TYPE)
IS
Begin
```

```
        Insert    INTO U    SER_data (    "User_Id", "    Username", "    Created_By",
      "Created_Date")
        Values (userid, username,createdby,pdate);
        Commit;
End;
/
```

Once the procedure insertUSERdata created, we will execute it from PL/SQL block as shown below.

Example:

```
Begin
        insertUSERdata(201,'Het','scott',SYSDATE);
End;
/
```

Execute the above PL/SQL block and check the table records.

II. UPDATE Statement

We will continue with the previously created user_data table. We will create a stored procedure which will accept 2 IN parameters and update the username field based on the provided userId.

Example:

```
Create or Replace Procedure updateUSERdata(
                userid IN USER_data.USER_ID%TYPE,
                newusername IN USER_data.USERNAME%TYPE)
IS
Begin
        Update  USER_data SET  Username =  newusername where  USER_ID =
      userid;
        Commit;
End;
```

```
/
```

Once the procedure updateUSERdata created, we will execute it from PL/SQL block as shown below.

Example:

```
Begin
        updateUSERdata(201,'Mansi');
End;
/
```

Execute the above PL/SQL block and check the table records.

III. DELETE Statement

We will continue with the previously created user_data table. We will create a stored procedure which will delete the record based on the provided userId.

Example:

```
Create or Replace Procedure deleteUSERdata(userid IN
USER_data.USER_ID%TYPE)
IS
Begin
        Delete USER_data where USER_ID = userid;
        Commit;
End;
/
```

Once the procedure deleteUSERdata created, we will execute it from PL/SQL block as shown below.

Example:

```
Begin
        deleteUSERdata(201);
End;
```

```
/
```

Execute the above PL/SQL block and check the table records.

2.4.2 DELETING A STORED PROCEDURE

To delete a stored procedure we have to fire following command.

Example:

- Drop procedure updateUSERdata;

Above code deletes the procedure updateUSERdata.

2.5 FUNCTION BASICS

A stored function is same as a procedure, except that it returns a value. Create Function command is used to create a stored function.

Syntax:

```
Create [OR Replace] Function function_name
[( parameter_1 [IN] [OUT] parameter_data_type_1,
parameter_2 [IN] [OUT] parameter_data_type_2,…
parameter_N [IN] [OUT] parameter_data_type_N )]
RETURN return_datatype
IS | AS
— declaration_statements
BEGIN
— executable_statements
return {return_data_type};
[EXCEPTION
— the exception-handling statements]
END [function_name];
```

Where,

1. The function_name is the name given to the PLSQL function.

2. The parameter_name is the name of the parameter that we are passing to the function.

3. The parameter_data_type is the data type of the parameter that we are passing to the PLSQL function.

4. Every Oracle PL/SQL function must have a Return statement in the code execution part.

The RETURN specified in the header part of the oracle PL/SQL function specifies the data-type of the value returned by the function.

2.5.1 PARAMETER PASSING TO A FUNCTION

There are 3 ways of passing parameters to PLSQL Function:

a. IN

b. OUT and

c. IN OUT

- **IN:** This is the default style of parameter in PLSQL function. This provides same functionality as of Stored Procedure.
- **OUT:** The OUT parameter returns the values to the calling subprogram or subroutines. This provides same functionality as of Stored Procedure.
- **IN OUT:** In this mode the actual parameter is passed to the PL/SQL function with initial values and then within the PL/SQL function the value of the parameter may get changed or reassigned. The IN OUT parameter is finally returned to the calling subroutine. This provides same functionality as of Stored Procedure.

The block structure of a PL/SQL function is same as those of a PL/SQL Anonymous Block. Anonymous Block doesn't have Create or Replace Function, the parameters section of code and the Return Clause.

To understand functions we will use the previously created table named 'employee'. Now suppose we want to create a function that shows us the name of an employee whenever we pass employee_id as parameter.

Example:

```
Create or Replace Function get_employee_name (emp_no IN number)
```

```
        RETURN varchar2
 IS
          emp_name varchar2(100);
Begin
           Select employee_name into emp_name
           From employee
           Where employee_id = emp_no;
        Return emp_name;
End get_employee_name;
/
```

Once the get_employee_name function created, we will execute it from PL/SQL block as shown below.

- **Calling Function**

We can call an Oracle PL/SQL Function two ways.

I. Using Oracle SQL SELECT statement

We can call the above PL/SQL function using an SQL SELECT statement shown below and check the output.

- Select get_employee_name (101) from dual;

Now, suppose if we change the employee_id passed to the function then we will get the name of another employee.

II. Using Oracle Anonymous Block

Second way to call function is to create an Anonymous block. Here we will create an anonymous block to call the get_employee_name PLSQL function.

Example:

```
Declare
          First_Name varchar2(30);
```

```
            Second_Name varchar2(30);
            Third_Name varchar2(30);
Begin
            First_Name := get_employee_name(101);
            Second_Name := get_employee_name(102);
            Third_Name := get_employee_name(103);

            dbms_output.put_line(First_Name);
            dbms_output.put_line(Second_Name);
            dbms_output.put_line(Third_Name);
End;
```

When we execute the above Oracle SQL Anonymous Block we will get three names as the output.

2.5.2 DELETING FUNCTION

To delete a function we have to use drop function command.

Syntax:

- Drop function <function-name>;

Example:

- Drop function get_employee_name;

Above code has deleted the function get_employee_name.

- **Check Your Progress**

1) What is procedure and function in PLSQL?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

2) Where the Pre_defined_functions are stored?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

3) Write the code for calling functions and procedures in a PLSQL block.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

4) Write any five inbuilt String function.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

5) State the similarities between Procedure and Function.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

6) Differentiate between Procedure and Function.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

2.6LET US SUM UP

In this chapter, we have learned PL/SQL subprograms. We have learned to create Procedure and different ways of calling it. We have also discussed to create Function and ways of calling it. We also learnt parameter passing and returning values from subprograms. In PLSQL stored procedure and function plays a very important role for passing and manipulating data records very efficiently and effectively.

2.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. A Procedure is a subprogram block consists of a group of PL/SQL statements while function is an independent PL/SQL subprogram.

2. Pre_defined_functions are stored in the standard package called “Functions, Procedures
and Packages”.

3. Function is called as a part of an expression:

Example: squr:=count_sqr('10');

Procedure is called as a statement in PL/SQL:

Example: count_salary('201');

4. Following are the five inbuilt String function:

I. INSTR(maintext, string, start, occurance): It gives the position of particular text in the given string.

Where,

maintext is main string,

string is text that need to be searched,

start indicates starting position of the search (optional),

accordance indicates the occurrence of the searched string (optional).

Example:

Select INSTR('Gujarat,'a',2,1) from dual;

Output: 4

II. UPPER (string): It returns the uppercase of the provided string.

Example:Select upper('baou') from dual;

Output: BAOU

III. LOWER (string): It returns the lowercase of the provided string.

Example:Select upper('BAOU') from dual;

Output: baou

IV. INITCAP (string): It returns the given string with the starting letter in upper case.

Example:Select ('gujarat vidyapith') from dual;

Output: Gujarat Vidyapith

V. LENGTH (text) Returns the length of the given string.

Example:Select LENGTH ('BAOU') from dual;

Output: 4

5. Both can be called from other PL/SQL blocks.

If the exception raised in the subprogram is not handled in the subprogram exception handling section, then it will propagate to the calling block.

Both can have as many parameters as required.

Both are treated as database objects in PL/SQL.

6. Following table shows the difference between Procedure and Function:

Procedure	Function
It is used to a execute certain process	It is used mainly to a execute certain calculations
It can't be called in Select statement	A Function without DML statements can be called in Select statement
It uses Out parameter to return the value	It uses Return to return the value
It is not mandatory to return the value from procedure	It is mandatory to return the value from function
Return will exit the control from subprogram.	Return will exit the control from subprogram along with returning the value
Return datatype is not required to be specified at the time of procedure creation	Return datatype is mandatory to specify at the time of function creation

2.8 ASSIGNMENTS

1. Define stored Procedure. Explain the characteristics of stored Procedure.
2. Define function. Explain the characteristics of functions.
3. Explain various Parameters of PLSQL subprograms.
4. Create a procedure that takes the pnum, pname as input and insert it to the 'tblPerson' table of the database.

5. Create a function that takes the number as input and returns the cube as output.

2.9 Further Reading

1. Advanced PL/SQL Programming: The Definitive Reference by Boobal Ganesan
2. SQL/PLSQL,TheProgrammingLanguageofORACLE,BPBPublicationbyIvan.
3. Introduction to Database Systems, 4th Edition, C. J. Date, Narose Publishing.

Unit 3:Package and Trigger

3

Unit Structure

3.1. Learning Objectives & Outcomes

3.2. Introduction

3.3. Package Component

3.4. Package Implementation

3.5. Trigger

3.6. Levels of Trigger

3.7. User

3.8. Let Us Sum Up

3.9. Check your progress: Possible Answers

3.10. Assignments

3.11. Further Reading

3.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this unit is to make the students,

• To learn and understand trigger and Package concepts

• To define, declare and initialize trigger on various kind of events

• To learn and initialize package and use it

• To learn the concept of Users and their roles

Outcome:

At the end of this unit,

• Students will be able to declare, initialize and write trigger based on various kinds of events

• Students will be able to define package and access that package

• Students will be able to create and remove user, grant and revoke privileges

3.2 INTRODUCTION

A Package is collection of objects. It contains procedures, functions, variables and SQL statements created as a single unit. A package consists of two parts, Package Specification or package header and Package Body.

Package Specification works as an interface to the package. Declaration of types, variables, constants, exceptions, cursors and subprograms is made in Package specifications. Package specification does not allow any code statements. Package body is the platform to provide implementation for the subprograms.

Package delivers various Advantages like,

- It allows user to group together related objects, types and subprograms as a PL/SQL module.
- If package contains a procedure and when a procedure is called first time, entire package is loaded. This is expensive with respect to resources. But it takes less response time for queries for subsequent calls.
- Package allows us to create types, variable and subprograms that are private or public

Items declared within package body are known as private. They are only accessed within the package. While items declared within package specification is public and available outside the package.

3.3 PACKAGE COMPONENT

Package component consists of two parts.

3.3.1 PACKAGE SPECIFICATION

The syntax for the package specification is as follows.

Syntax:

```
CREATE [OR REPLACE] PACKAGE package_name
        [ AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
          [Definitions of public TYPES
          ,Declarations of public variables, types, and objects
          ,Declarations of Exceptions
          ,Pragmas
          ,Declarations of Cursors, Procedures, and Functions
          ,Headers of Procedures and Functions]
END [package_name];
```

3.3.2 PACKAGE BODY

The syntax for the package body is as follows:

Syntax:

```
CREATE [OR REPLACE] PACKAGE BODY package_name
{ IS | AS }
           [Definitions of private TYPEs
          ,Declarations of private variables, types, and objects
          ,full definitions of Cursors
          ,full definitions of Procedures and Functions]
```

```
        [BEGIN
          sequence_of_statements
        [EXCEPTION
          Exception_handlers ] ]
 END [package_name];
```

Package body is not required if the package specification contains only types, constants, variables, exceptions. This type of packages only contains global variables that will be used by subprograms or cursors.

3.4 Package Implementation

Now we will discuss the implementation of package. First of all, we will start with simple example as follows:

Example 1: In the below code, first we are creating a package specification with two stored procedure one to find the maximum number and another to find the cube of the given number.

Package Specification:

```
Create or Replace Package PackageTest as
   procedure findMaximum(num1 IN number, num2 IN number);
   procedure findCube (num IN number);
end PackageTest;
/
```

Once we execute above code it will create a package specification named 'PackageTest' (the body is not created yet).

Package Body:

Now consider the following code:

```
Create or Replace Package body PackageTest as
```

```
            procedure findMaximum(num1 IN number, num2 IN number) is
            begin
                 if (num1 > num2) then
                         dbms_output.put_line (num1|| ‘ is greater than ‘|| num2);
                 else
                         dbms_output.put_line (num2|| ‘ is greater than ‘|| num1);
                 end if;
           end;
            procedure findCube(num IN number) is
           begin
               dbms_output.put_line (‘Cube of the number ‘|| num || ‘ is ’|| (num * num *
         num));
            end;
end PackageTest;
/
```

When we execute the above code it will create the package body for the previously created package specification. All the members in the package body must match with all the declarations within the package specification. We have to make sure that both package specification and package body gets stored in the database.

To execute package we have to use the command ‘execute’ followed by the “packagename.sub-programname”. To execute the above created package from SQL prompt the following command will be used.

- Execute PackageTest.findcube(15);
- Execute PackageTest.findMaximum(15,25);

Both of the above execution will return the respective output.

Example 2:

Now we will create a package to interact with a database. Before creating a package we will create tables named Employee and Department to be accessed in package as shown below.

- Create table employee(eno number(3) primary key,ename varchar2(15),salary number(7,2), deptno number(3) references department);
- Create table department(deptno number(3) primary key, deptname varchar2(15));

After creating both the tables insert few records in both the tables.

After inserting records into the tables we will create package to access both the tables in it.

Package Specification:

```
Create or Replace Package PackageDBAccess as
   procedure dispEmprecord;
   procedure dispDeptrecord;
end PackageDBAccess;
/
```

Package Body:

```
Create or Replace Package body PackageDBAccess AS
           Procedure dispEmprecord as
              Cursor cursor_emprec is
                 select ename, salary from employee;
          Begin
              dbms_output.put_line ('Name' || '        ' || 'Salary');
             for record_emp in cursor_emprec
             loop
                      dbms_output.put_line (record_emp.ename || '        ' ||
             record_emp.salary);
             end loop;
          End;
          Procedure dispDeptrecord as
              Cursor cursor_deptrec is
                 select deptno,deptname from department;
```

```
        Begin
           dbms_output.put_line ('DeptNo' || '          ' || 'DeptName');
           for                record_dept              in              cursor_deptrec
           loop
                 dbms_output.put_line (record_dept.deptno || '        ' ||
                   record_dept.deptname);
           end                                                                  loop;
        End;
End PackageDBAccess;
/
```

Above block of code will successfully create a package body.

Package Execution

To execute each of these procedures separately, we can use the following command as shown below.

- Execute PackageDBAccess. dispemprecord;
- Execute PackageDBAccess. dispdeptrecord;

When we execute both the above statements it will display both table records.

3.4.1 ALTERING PACKAGE

Sometime we need to modify the package code. So, after updating the code we have to just recompile the package body.

Package Alter Syntax is:

- Alter Package <package_name> Compile Body;

3.4.2 DELETING PACKAGE

To delete the package we have to use package Drop command.

Package Drop Syntax is:

- Drop Package <package_name>;

3.5 TRIGGERS

A database trigger is a stored procedure associated with a database table, view or event. The trigger can be invoked once, when some event occurs. It may occur many times, once for each row affected by an Insert, Update or Delete statement. The trigger can be invoked before the event to prevent unexpected operations. The executable part of a trigger can contain procedural statements and SQL statements. The stored procedure and functions have to be called explicitly while the database triggers are executed or called implicitly whenever the table is affected by any DML operations.

We can write triggers that will be invoked whenever one of the following operations occurs:

- DML commands (Insert, Update, Delete) on a particular table or view issued by any user.
- DDL commands (Create or Alter primarily) issued either by a particular schema/user or by any schema/user in the database.
- Database events such as logon/logoff, errors or startup/shutdown, issued either by a particular schema/user or by any schema/user in the database

➢ Uses of Triggers

1. Trigger allows enforcing business rules that can't be defined by using integrity constants.
2. Trigger enables us to gain strong control over the security.
3. Using trigger we can also collect statistical information on the table access.
4. Using triggers we can prevent invalid transaction.

3.5.1 TYPES OF TRIGGERS

Trigger type depends on the type of triggering operation and by the level at which the trigger is executed. Triggers are of Two Types.

3.5.1.1 Row Level Triggers

A row trigger is triggered each time a row in the table is affected by the triggering statement. For example, if an update statement updates multiple rows of a table, a row trigger is triggered once for each row affected by the update statement. If the triggering statement affects no rows, the trigger is not executed. Row triggers should be used when some processing is required whenever a triggering statement affects a single row in a table. Row level triggers are created using the “For Each Row” Clause in the Create Trigger statement.

3.5.1.2 Statement Level Triggers

A statement level trigger is triggered once on behalf of the triggering statement, independent of the number of rows the triggering statement affects (even if no rows are affected). Statement triggers should be used when a triggering statement affects rows in a table but the processing required is completely independent of the number of rows affected. Statement level triggers are the default trigger created via Create Trigger statement.

Syntax:

```
CREATE [OR REPLACE ] TRIGGER Trigger_Name
        {BEFORE | AFTER | INSTEAD OF }
        {INSERT [OR] | UPDATE [OR] | DELETE}
        [OF col_name]
        ON table_name
        [REFERENCING OLD AS o NEW AS n]
        [FOR EACH ROW]
        WHEN (condition)
BEGIN
         --- SQL statements
 END;
```

Explanation:

- CREATE [OR REPLACE] TRIGGER trigger_name : This creates a trigger with the given name or overwrites an existing trigger with the same name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies at what time the trigger get fired. i.e before or after updating a table. Before means before compiling the statement the trigger will be fired, after means after the compilation the trigger

will be fired. INSTEAD OF is used to create a trigger on a view. Before and after cannot be used to create a trigger on a view.

- {INSERT [OR] | UPDATE [OR] | DELETE} : This determines the triggering event. There are more than one triggering events that can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
- [OF col_name] : This clause is used with update triggers. This clause is used when we want the trigger to fire only when a specific column is updated.
- [ON table_name] : This clause specifies the name of the table or view to which the trigger is associated.
- [REFERENCING OLD AS o NEW AS n] : This clause is used to reference the old and new values of the data being changed. By default, we reference the values as : old.column_name or : new.column_name. We cannot reference old values when inserting a record, or new values when deleting a record because they do not exist.
- [FOR EACH ROW] : This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire SQL statement is executed (i.e. statement level Trigger).
- WHEN (condition) : This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the specified condition.

3.5.1.3 INSTEAD OF Trigger

This type of trigger enables us to stop and redirect the performance of a DML statement. This type of trigger helps us in managing the way we write to non-updatable views. Sometimes, the INSTEAD OF triggers are also seen inserting, updating or deleting rows in designated tables that are otherwise unrelated to the view.

3.5.1.4 Compound Triggers

These are multi-tasking triggers that work as both statement as well as row-level triggers when the data is inserted, updated or deleted from a relation.

3.5.2 DML TRIGGERS

These triggers are executed before or after we perform any DML operations on a table. When we create a trigger, the trigger definition is stored in the database, which is identified with the trigger name. The code in the trigger is processed when we apply any command on the database or table.

- **Statement Level Triggers:**

Example 1: Create a Trigger, which displays a message whenever we insert a new row in to Employee table.

```
Create or replace trigger instrigger before insert on Employee
Begin
            dbms_output.put_line('one record inserted successfully.....');
End;
/
```

Example 2. Create a Trigger, Which displays a message whenever we update an existing row in the tableEmployee**.**

```
Create or replace trigger updtrigger before update on Employee
Begin
            dbms_output.put_line('one record updated successfully.....');
End;
/
```

Example 3. Create a Trigger, which displays a message whenever we delete a row from the table Employee.

```
Create or replace trigger deltrigger before delete on Employee
Begin
            dbms_output.put_line('record(s) deleted successfully.....');
End;
/
```

- **Row Level Triggers:**

Example 1. Create a Trigger, which displays a message whenever we insert a new row into a tableEmployee**.**

```
Create or replace trigger instrigger before insert on Employee
for each row
Begin
        dbms_output.put_line(:new.id||' record inserted successfully.....');
End;
/
```

Example 2. Create a trigger, which displays a message whenever we update a row in the table Employee.

```
Create or replace trigger updtrigger before update on Employee
for each row
Begin
        dbms_output.put_line(:old.id||' record updated to '||:new.id);
End;
/
```

Example 3. Create a Trigger, which displays a message whenever we delete a row from the table Employee.

```
Create or replace trigger deltrigger after delete on Employee
for each row
Begin
        dbms_output.put_line(:old.id||' record deleted successfully.....');
End;
/
```

3.5.3 DDL TRIGGERS

Example 1. Create a Trigger, which displays an error message whenever we create a new table.

```
Create or replace trigger restrict_CreateTable
```

```
before create on schema
begin
        raise_application_error(-20001,'CREATE Table not Permitted');
end;
/
```

As we can see that the above code creates a trigger restrict_CreateTable. Now when we try to create a table named test it will not allow us to do so.

Example 2. Create a Trigger, which will display an error message whenever we try to drop any table. Now create one table named Test as shown below.

- Create table Test(tno number(3),tname varchar2(20));

```
Create or replace trigger restrict_DropTable
before drop on schema
begin
        raise_application_error(-20001,'DROP Table not permitted');
end;
/
```

After the above block of code gets executed it will create a trigger restrict_DropTable. Now try to drop the previously created table Test and check the output.

Example 3. Create a Trigger, which will display an error message whenever we try to alter any table**.**

```
Create or replace trigger restrict_AlterTable
before alter on schema
begin
        raise_application_error(-20001,'ALTER Table not permitted');
end;
  /
```

After the above block of code gets executed it will create a trigger restrict_AlterTable. Now try to alter the previously created table Test and check the output.

Example 4. Create a Trigger, which displays an error message whenever we try to truncate any table**.**

```
Create or replace trigger restrict_TruncateTable
before truncate on schema
begin
        raise_application_error(-20001,'TRUNCATE table not Permitted');
end;
/
```

After the above block of code gets executed it will create a trigger restrict_TruncateTable. Now try to truncate the previously created table Test and check the output.

3.6 LEVELS OF TRIGGER

Level of trigger can be categorized as follows.

3.6.1 BEFORE INSERT TRIGGER

A Before Insert trigger means the trigger will be fired before the insert operation is executed.

Syntax:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
BEFORE INSERT
          ON table_name
          [ FOR EACH ROW ]
        DECLARE
          -- variable declarations
BEGIN
```

```
        -- trigger code
      EXCEPTION
        WHEN ...
        -- exception handling
END;
```

Suppose we have a table named Customer_Order created as follows:

```
Create Table Customer_Order
      (Custorder_id number(5),   Ordquantity number(4),
      cost_per_Orditem number(6,2),   total_Ordcost number(8,2),
  ord_date date,   Ordcreated_by varchar2(10) );
```

After creating the table, we can then use the Create Trigger statement to create a Before Insert Trigger as follows:

Example:

```
Create or Replace Trigger Before_InsertData
Before Insert ON Customer_Order
      For Each Row
      Declare
      u_name varchar2(10);
Begin
        Select user INTO u_name from dual;
        -- Update ord_date field with current system date
        :new.ord_date := sysdate;
-- Update Ordcreated_by field to the username of the person performing the
Insert
        :new.Ordcreated_by := u_name;
      dbms_output.put_line('The Trigger Executed Successfully');
End;
```

```
/
```

Once the trigger is created insert following records into the table. When we insert the records the trigger will be invoked implicitly.

- insert into Customer_Order values(1,12,5,60,'28-march-19','vinod');
- insert into Customer_Order values(2,5,15,75,'28-march-19','mukesh');

By observing the above execution, we can say that when we have inserted the records with date and user '28-march-19','vinod' & '28-march-19','mukesh' respectively; the created trigger will fire implicitly on Customer_Order table and replace the date and user values as per the trigger body.

Note: The values in Ord_Date and OrdCreated_By columns may be different for you as they depend on system date and user logged in.

3.6.2 AFTER INSERT TRIGGER

An After Insert Trigger means that the trigger will be fired after the insert operation is executed.

Syntax:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
        AFTER INSERT
          ON table_name
          [ FOR EACH ROW ]

        DECLARE
          -- variable declarations

        BEGIN
          -- trigger code

        EXCEPTION
          WHEN ...
```

```
        -- exception handling

END;
/
```

Example:

Suppose we have a table named Customer as follows:

```
Create Table Customer
(emp_id number(4), emp_name varchar2(30), creation_date date, created_by
varchar2(30) );
```

We will also create a duplicate table of 'Customer' table as 'Duplicate_Customer' using the code below:

```
Create Table Duplicate_Customer As (select * from Customer);
```

At this moment we have not inserted any data in 'Customer' and 'Duplicate_Customer' tables. Now, create a trigger on 'Customer' table so that whenever we will enter any customer record in the 'Customer' table the same record also gets stored in 'Duplicate_ Customer' table.

Trigger:

```
Create or Replace Trigger After_InsertData_trigger
After Insert
      ON Customer
      For each row
      Declare
        creator_name varchar2(30);
        creation_date date;
Begin
```

```
        --Getting the name of the current logged in User
          Select User INTO creator_name   From dual;
        --setting system date in creation_date
          creation_date := sysdate;
        --Inserting data into the Duplicate_Customer table
          Insert into Duplicate_Customer
          Values (   :new.emp_id ,    :new.emp_name ,     c    reation_date ,
        creator_name);
End;
/
```

Here w e ha ve c reated a P L/SQL After Insert T rigger named ‘After_InsertData_trigger’ which will insert a record in the ‘Duplicate_Customer’ table as soon as insert operation is performed on ‘Customer’ table.

Let’s insert a row in ‘Customer’ table as:

- Insert Into Customer Values (1, ‘himanshu’,sysdate,’vinod’);

After executing above Insert statement, we can query on both the tables and check the output.

Here using the PL/SQL After Insert Trigger we can see that in the ‘Duplicate_Customer’ table a record got inserted as soon as we inserted a record in ‘Customer’ table.

We c an al so create t rigger f or bef ore upd ate, af ter upd ate, b efore d elete and af ter delete operations.

3.6.3 DROP TRIGGER

After c reating a t rigger i n O racle, w e m ight f ind t hat w e need t o r emove i t f rom t he database. We can do this with the Drop Trigger statement.

Syntax:

- Drop Trigger Trigger-Name;

Example:

- Drop trigger After_InsertData_trigger;

3.6.4 ENABLE-DISABLE TRIGGER

Whenever we need to disable the trigger, we can do this with the Alter Trigger statement.

Example:

- ALTER Trigger Before_Insert_Trigger DISABLE;

Above statement uses the Alter Trigger statement to disable the trigger called Before_Insert_Trigger.

- **Disable all Triggers on a Table**

We can disable all triggers associated with a table at the same time using the Alter Table statement with the Disable All Triggers option. For example, to disable all triggers defined for the Customer_Order table, we can write the following command.

Syntax:

- Alter table table_name Disable All Triggers;

- **Enable a Trigger**

Sometimes we want to enable trigger on a table which is disabled earlier. We can do this with the help of Alter Trigger statement.

Syntax:

- ALTER TRIGGER trigger_name ENABLE;

Example:

- ALTER TRIGGER orders_before_insert ENABLE;

This example uses the Alter Trigger statement to enable the trigger called orders_before_insert.

- **Enable all Triggers on a Table**

We can enable all triggers associated with a table at the same time using the Alter Table statement with the Enable All Triggers option. To enable all triggers defined for the Customer_Order table, enter the following command.

Syntax:

- Alter Table table_name Enable All Triggers;

Example:

- Alter Table Customer_Order Enable All Triggers;

3.7 USER

To create a user, simply issue the Create User command to generate a new account.

3.7.1 CREATING A USER

Example:

- Create User Ved Identified By rdbms;

Here we have simply created a Ved account that is identified or authenticated by the rdbms password.

- **Privileges and Roles**

Privilegesdefines the access rights provided to a user on a database objects. There are two types of privileges:

I. System Privileges: This privilege allows user to create, alter, or drop database elements.

II. Object Privileges: This privilege allows user to execute, select, insert, or delete data from database objects to which the privileges apply.

Roles are the collection of privileges or access rights. In case of many users in a database it becomes complex to grant or revoke privileges to the users. So, if we define roles we can automatically grant/revoke privileges.

Data Control Language (DCL) commands are used to enforce database security in a multiple database environment. Two types of DCL commands used are Grant and Revoke. Database Administrator's or owner's of the database object can provide or remove privileges on a database object.

3.7.2 GRANT COMMAND

SQL Grant command is used to provide access or privileges on the database objects to the users. The **syntax** for the GRANT command is:

- GRANT privilege_name ON object_name TO { user_name |PUBLIC |

role_name} [with GRANT option];

Where,

- privilege_name is the access right or privilege granted to the user.
- object_name is the name of the database object like table, view etc.
- user_name is the name of the user to whom an access right is being granted.
- Public is used to grant rights to all the users.
- With Grant option allows users to grant access rights to other users.

In create user section, we have Ved account created, we can now start adding privileges to the account using the GRANT statement. GRANT is a very important and powerful command with many possible options. Generally, we first want to assign privileges to the user through connecting the account to various roles.

Syntax:

- GRANT<privilege> to <user>

Example:

- Grant Connect to Ved;

To allow your user to login, we need to give it the create session privilege as shown below:

- Grant create session to Ved;

We can give many system privileges in one command also. Grant these to Ved by chaining them together as shown below:

- Grant create table, create view, create procedure, create sequence to Ved;

In newer versions of oracle it is not necessary but some older version may require that we manually assign the access rights to the new user to a specific schema and database tables.

For example, if we want our Ved user to have the ability to perform Select, Update, Insert and Delete operation on the student table, we might execute the following GRANT statement:

- Grant select, insert, update, delete on schema.student to Ved;

This ensures that Ved can perform the four basic operation for the student table that is part of the database schema.

3.7.3 REVOKE COMMAND

The revoke command removes user access rights or privileges to the database objects. The syntax for the REVOKE command is:

- REVOKE privilege_name ON object_name FROM{ User_name | PUBLIC | Role_name}

For example to revoke select, update, insert privilege granted to Ved then give the following statement.

- revoke select, update, insert on employee from Ved;

To revoke update statement on employee granted to public then give the following command.

- revoke update on employee from public;

- **Revoking System Privileges and Roles:**

We can revoke system privileges or roles using the SQL command revoke. Any user with the admin capacity for a system privilege or role can revoke the privilege or role from any other database user. The grantor does not have to be the user that originally granted the privilege or role. The following statement revokes the create table System Privilege from Ved:

- Revoke create table from Ved;

- **Revoking Object Privileges and Roles:**

We can revoke object privileges using the SQL command revoke. To revoke an object privilege, the revoker must be the original grantor of the object privilege being revoked. For example, assuming you are the original grantor, to revoke the select and insert privileges on the employee table from the users Ved and Shrey, you have to issue the following command:

- Revoke select, insert on employee from Ved, Shrey;

- **Revoking Column Selective Object Privileges:**

Users can grant specific column level insert, update and references privileges for tables and views. But they cannot revoke column specific privileges with a similar revoke statement. For that, the grantor must first revoke the object privilege for all columns of a table or view, and then regrant the column specific privileges.

For example, assume that role Computer_Science is granted the update privilege on the deptId and dname columns of the table dept. To revoke the update privilege on just the deptId column, we have to issue the following two commands:

- Revoke update on dept from Computer_Science;
- Grant update (dname) on dept to Computer_Science;

The revoke statement revokes update privilege on all columns of the dept table from the role Computer_Science. The grant statement regrants update privilege on the dname column to the role Computer_Science.

3.7.4 DROP USER

The DROP USER command is used to remove a user from the oracle database and remove all objects owned by that user.

Syntax:

- DROP USER user_name [CASCADE];

Where:

user_name: It specifies the name of the user to remove from the Oracle database.

CASCADE: It is optional. It specifies that if user_name owns any objects (i.e. tables or views in its schema), we must specify CASCADE to drop all of these objects.

Example:

If the user does not own any objects in its schema, we can execute the following DROP USER statement:

- DROP USER Ved;

Above code will drop the user called Ved. This DROP USER command will only run if Ved does not own any objects in its schema.

If Ved did own objects in its schema, we will need to run the following DROP USER command:

- DROP USER Ved CASCADE;

This DROP USER statement will remove the user Ved, drop all objects (i.e. tables and views) owned by Ved, and all referential integrity constraints on Ved's objects will also be dropped.

- **Check Your Progress**

1) What is Trigger?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

2) When do we use triggers?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

3) What is INSTEAD OF triggers?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

4) Differentiate between execution of triggers and stored procedures.

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

5) Write the objects that PL/SQL package may contain.

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

6) What is PL/SQL packages? State two different parts of the PL/SQL packages.

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

7) What do you mean by privileges and Grants?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

3.8 LET US SUM UP

In this unit we have discussed package and trigger. Package allows us to bundle all the objects like function, procedure within it and later we can execute them either directly or from other subprograms. We also learnt that the trigger can be invoked whenever an event occurs. Event may be an Insert, Update or Delete statement. Throughout Trigger discussion we observed that it helps us in enforcing business rules that can't be defined by using integrity constants. We can generate statistical data using trigger about the table access. Through trigger we can prevent invalid transaction from execution. So, both package and trigger objects of PLSQL allows programmer a wide scope in writing sub programs. At last we have learnt the creation of user, granting roles and privileges to users and removing the users.

3.9 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Trigger is a database object, executes automatically in response to some events on the tables or views. It is used to maintain the integrity constraint to the database objects.

2. The word 'Trigger' means to activate. Triggers are mainly required for the following goals:

- To maintain complex integrity constraints on the database tables
- To audit table information by recording the changes
- To signal other program actions when changes are made to database table
- To enforce complex business rules
- To preventing invalid transactions

3. The INSTEAD OF triggers are written especially for updating views, which is not possible to modify directly through SQL DML statements.

4. Stored procedure is executed explicitly by issuing procedure call statement from another block while trigger is executed implicitly whenever any triggering event like any DML operation happens.

5. A PL/SQL package contains;

- PL/SQL table and record TYPE statements
- Procedures and Functions
- Cursors
- Variables and constants
- Exception and pragmas for associating an error number with an exception

6. PL/SQL package is a schema that groups functions, cursors, stored procedures and variables in one place. PL/SQL packages have the following two parts:

I. Specification part: This part specifies the part where the interface to the application is defined.

II. Body part: Body part specifies the implementation of the specification is defined.

7. Privileges are the rights to execute SQL commands. Grants are assigned to the object so that objects can be accessed accordingly. Grants can be assigned by the owner or creator of an object.

3.10 ASSIGNMENTS

1. Explain the uses of database trigger?
2. Explain 3 basic parts of a trigger.
3. What are the benefits of PL/SQL packages?
4. Explain the difference between Triggers and Constraints?
5. Explain types of triggers supported by PL/SQL with example.
6. Write a trigger that may execute after deleting a record from the table.
7. Define User, role and privileges.
8. Explain Grant and Revoke command with proper example.

3.11 Further Reading

1. Advanced PL/SQL Programming: The Definitive Reference by Boobal Ganesan
2. SQL/PLSQL,TheProgramming Language of ORACLE,BPB Publication by Ivan.
3. Introduction to Database Systems, 4th Edition, C. J. Date, Narose Publishing.
4. http://beginner-sql-tutorial.com/sql-grant-revoke-privileges-roles.htm

Unit 4:Managing User Privileges & Roles and User Profile

4

Unit Structure

4.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this chapter is to make the students,

• To understand User Role

• To learn about Privileges

• To understand User Profile.

Outcome:

At the end of this unit,

• Students will be able to understand User Role and Privileges.

• Students will be able to create User Defined Role and assign it to the Users.

• Students will understand difference between System Privileges and Schema Objects Privileges.

• Students will be able to create User Profile.

4.2 INTRODUCTION

Roles, on the other hand, are created by users (usually administrators) and are used to group together privileges or other roles. They are a means of facilitating the granting of multiple privileges or roles to users. A user privilege is a right to execute a particular type of SQL statement, or a right to access another user's object.

Each role and user has its own unique security domain. A role's security domain includes the privileges granted to the role plus those privileges granted to any roles that are granted to the role.

A user's security domain includes privileges on all schema objects in the corresponding schema, the privileges granted to the user, and the privileges of roles granted to the user that are *currently enabled*. A role can be simultaneously enabled for one user and disabled for another. A user's security domain also includes the privileges and roles granted to the user group PUBLIC. The SESSION_ROLES view shows all roles that are currently enabled.

In some environments, you can administer database security using the operating system. The operating system can be used to manage the granting (and revoking) of database roles and to manage their password authentication. This capability is not available on all operating systems.

This chapter describes management of different SQL concepts as follows:

- User Roles
- Privileges
- User Profiles.

4.3 User Role

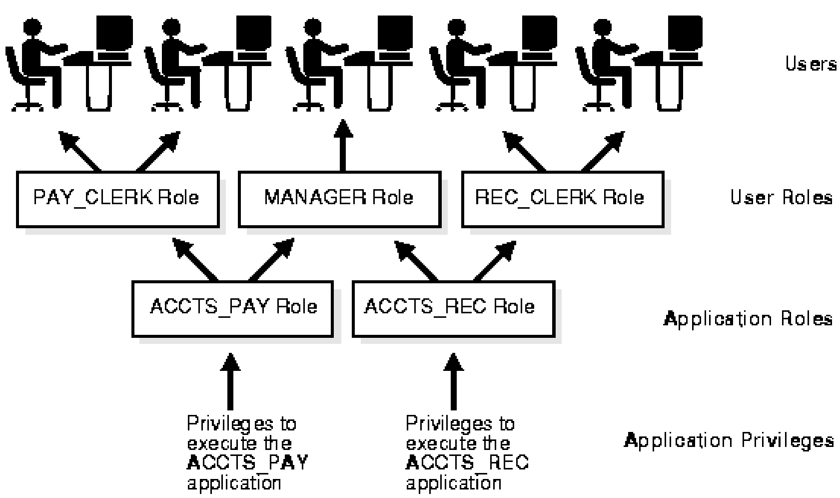
Oracle provides for easy and controlled privilege management through roles. Roles are named groups of related privileges that you grant to users or other roles. Roles are designed to ease the administration of end-user system and schema object privileges. However, roles are not meant to be used for application developers, because the privileges to access schema objects within stored programmatic constructs need to be granted directly.

These properties of roles allow for easier privilege management within a database:

Reduced privilege administration	Rather than granting the same set of privileges explicitly to several users, you can grant the privileges for a group of related users to a role, and then only the role needs to be granted to each member of the group.
Dynamic privilege management	If the privileges of a group must change, only the privileges of the role need to be modified. The security domains of all users granted the group's role automatically reflect the changes made to the role.
Selective	You can selectively enable or disable the roles granted to a user.

availability of privileges	This allows specific control of a user's privileges in any given situation.
Application awareness	The data dictionary records which roles exist, so you can design applications to query the dictionary and automatically enable (or disable) selective roles when a user attempts to execute the application by way of a given username.
Application-specific security	You can protect role use with a password. Applications can be created specifically to enable a role when supplied the correct password. Users cannot enable the role if they do not know the password.

In general, you create a role to serve one of two purposes: to manage the privileges for a database application or to manage the privileges for a user group.



Application Roles: You grant an application role all privileges necessary to run a given database application. Then, you grant the application role to other roles or to specific users. An application can have several different roles, with each role assigned a different set of privileges that allow for more or less data access while using the application.

User Roles: You create a user role for a group of database users with common privilege requirements. You manage user privileges by granting application roles and privileges to the user role and then granting the user role to appropriate users.

Database roles have the following functionality:

- A role can be granted system or schema object privileges.
- A role can be granted to other roles. However, a role cannot be granted to itself and cannot be granted circularly.
- Any role can be granted to any database user.
- Each role granted to a user is, at a given time, either enabled or disabled.
- An indirectly granted role (a role granted to a role) can be explicitly enabled or disabled for a user. However, by enabling a role that contains other roles, you implicitly enable all indirectly granted roles of the directly granted role.

Granting and Revoking Roles

You grant or revoke roles from users or other roles using the following options:
- The Grant System Privileges/Roles dialog box and Revoke System Privileges/Roles dialog box of Oracle Enterprise Manager
- The SQL commands GRANT and REVOKE

Roles can also be granted to and revoked from users using the operating system that executes Oracle, or through network services.

Any user with the GRANT ANY ROLE system privilege can grant or revoke any role (except a global role) to or from other users or roles of the database. Any user granted a role with the ADMIN OPTION can grant or revoke that role to or from other users or roles of the database.

Predefined Roles

The roles CONNECT, RESOURCE, DBA, EXP_FULL_DATABASE, and IMP_FULL_DATABASE are defined automatically for Oracle databases. These roles are provided for backward compatibility to earlier versions of Oracle and can be modified in the same manner as any other role in an Oracle database.

4.4 Privileges

A privilege is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include the right to

- connect to the database (create a session)
- create a table
- select rows from another user's table
- execute another user's stored procedure

You grant privileges to users so these users can accomplish tasks required for their job. Excessive granting of unnecessary privileges can compromise security. A user can receive a privilege in two different ways:

- You can grant privileges to users explicitly.
- You can also grant privileges to a role (a named group of privileges), and then grant the role to one or more users.

There are two distinct categories of privileges:

- **System privileges**
- **Schema object privileges**

A. System Privileges

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges. There are over 100 distinct system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations.

You can grant or revoke system privileges to users and roles. If you grant system privileges to roles, you can use the roles to manage system privileges System privileges are granted to or revoked from users and roles using either of the following:

- The Grant System Privileges/Roles dialog box and Revoke System Privileges/Roles dialog box of Oracle Enterprise Manager
- The SQL commands GRANT and REVOKE

Only users who have been granted a specific system privilege with the ADMIN OPTION or users with the GRANT ANY PRIVILEGE system privilege can grant or revoke system privileges to other users.

Because system privileges are so powerful, Oracle recommends that you configure your database to prevent regular (non-DBA) users exercising ANY system privileges (such as UPDATE ANY TABLE) on the data dictionary. In order to secure the data dictionary, ensure that the O7_DICTIONARY_ACCESSIBILITY initialization parameter is set to FALSE. This feature is called the **dictionary protection mechanism**.

B. Schema Object Privileges

A schema object privilege is a privilege or right to perform a particular action on a specific table, view, sequence, procedure, function, or package. Different object privileges are available for different types of schema objects.

Some schema objects (such as clusters, indexes, triggers, and database links) do not have associated object privileges; their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the ALTER ANY CLUSTER system privilege.

A schema object and its synonym are equivalent with respect to privileges; that is, the object privileges granted for a table, view, sequence, procedure, function, or package apply whether referencing the base object by name or using a synonym.

Schema object privileges can be granted to and revoked from users and roles. If you grant object privileges to roles, you can make the privileges selectively available. Object privileges for users and roles can be granted or revoked using the SQL commands GRANT and REVOKE, respectively, or the Add Privilege to Role/User dialog box and Revoke Privilege from Role/User dialog box of Oracle Enterprise Manger.

4.5 Managing User Role and Privileges

4.5.1. CREATE ROLE

You may wish to create a role so that you can logically group the users' permissions. Please note that to create a role, you must have CREATE ROLE system privileges.

You must give each role you create a unique name among existing user names and role names of the database. Roles are not contained in the schema of any user. In a database that uses a multibyte character set, Oracle recommends that each role name contain at least one single-byte character. If a role name contains only multibyte characters, then the encrypted role name and password combination is considerably less secure.

Syntax

CREATE ROLE <ROLE_NAME>
[NOT IDENTIFIED | IDENTIFIED {BY password | USING [schema.] package | EXTERNALLY | GLOBALLY }] ;

Where,
ROLE_NAME: The name of the new role that you are creating. This is how you will refer to the grouping of privileges.
NOT IDENTIFIED: It means that the role is immediately enabled. No password is required to enable the role.
IDENTIFIED: It means that a user must be authorized by a specified method before the role is enabled.
BY password: It means that a user must supply a password to enable the role.
USING package: It means that you are creating an application role - a role that is enabled only by applications using an authorized package.
EXTERNALLY: It means that a user must be authorized by an external service to enable the role. An external service can be an operating system or third-party service.
GLOBALLY: It means that a user must be authorized by the enterprise directory service to enable the role.

If both NOT IDENTIFIED and IDENTIFIED are omitted in the CREATE ROLE statement, the role will be created as a NOT IDENTIFIED role.

Example
CREATE ROLE DEMO_ROLE;
It will create New Role called DEMO_ROLE;

A. Grant TABLE Privileges to Role

Once you have created the role in Oracle, your next step is to grant privileges to that role.

Just as you granted privileges to users, you can grant privileges to a role. Let's start with granting table privileges to a role. Table privileges can be any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

Syntax

GRANT <PRIVILEGES> ON <OBJECT> TO <ROLE_NAME>;

Where,

Privileges: The privileges to assign to the role. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

Object: The name of the database object that you are granting privileges for. In the case of granting privileges on a table, this would be the table name.

Role_Name: The name of the role that will be granted these privileges.

Example

1. If you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called salesman to a role named DEMO_ROLE, you would run the following GRANT statement:

GRANT select, insert, update, delete ON salesman TO DEMO_ROLE;

2. You can also use the ALL keyword to indicate that you wish all permissions to be granted. **GRANT all ON salesman TO DEMO_ROLE;**

B. Revoke Table Privileges from Role

Once you have granted table privileges to a role, you may need to revoke some or all of these privileges. To do this, you can execute a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

Syntax

REVOKE <PRIVILEGES> ON <OBJECT> FROM <ROLE_NAME>;

Where,

Privileges: The privileges to revoke from the role. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

Object: The name of the database object that you are revoking privileges for. In the case of revoking privileges on a table, this would be the table name.

Role_Name: The name of the role that will have these privileges revoked.

Example

1. If you wanted to revoke DELETE privileges on a table called salesman from a role named DEMO_ROLE, you would run the following REVOKE statement:

REVOKE delete ON salesman FROM DEMO_ROLE;

2. If you wanted to revoke ALL privileges on the table called Salesman from a role named DEMO_ROLE, you could use the ALL keyword.

REVOKE all ON salesman FROM DEMO_ROLE;

4.5.2. GRANT ROLE TO USER

Now, that you've created the role and assigned the privileges to the role, you'll need to grant the role to specific users.

Syntax

GRANT <ROLE_NAME> TO <USER_NAME>;

Where,

Role_Name: The name of the role that you wish to grant.

User_Name: The name of the user that will be granted the role.

Example

1. GRANT DEMO_ROLE TO SCOTT;

This example would grant the role called DEMO_ROLE to the user named SCOTT.

A. Enable/Disable Role (Set Role Statement)

To enable or disable a role for a current session, you can use the SET ROLE statement. When a user logs into Oracle, all default roles are enabled, but non-default roles must be enabled with the SET ROLE statement.

Syntax

SET ROLE (ROLE_NAME [IDENTIFIED BY PASSWORD] | ALL [EXCEPT ROLE1, ROLE2, ...] | NONE);

Role_Name: The name of the role that you wish to enable.

IDENTIFIED BY password: The password for the role to enable it. If the role does not have a password, this phrase can be omitted.

ALL: It means that all roles should be enabled for this current session, except those listed in EXCEPT.

NONE: Disables all roles for the current session (including all default roles).

Example

SET ROLE DEMO_ROLE IDENTIFIED BY demo123;

This enable the role called DEMO_ROLE with a password of demo123.

B. Set role as DEFAULT Role

A default role means that the role is always enabled for the current session at logon. It is not necessary to issue the SET ROLE statement. To set a role as a DEFAULT ROLE, you need to issue the ALTER USER statement.

Syntax

ALTER USER <USER_NAME> DEFAULT ROLE (<ROLE_NAME> | ALL [EXCEPT ROLE1, ROLE2, ...] | NONE);

Where,

USER_NAME: The name of the user whose role you are setting as DEFAULT.

ROLE_NAME: The name of the role that you wish to set as DEFAULT.

ALL: It means that all roles should be enabled as DEFAULT, except those listed in EXCEPT.

NONE: Disables all roles as DEFAULT.

Example

ALTER USER scott DEFAULT ROLE DEMO_ROLE;

It would set the role called DEMO_ROLE as a DEFAULT role for the user named scott.

4.5.3. DROP ROLE

In some cases, it may be appropriate to drop a role from the database. The security domains of all users and roles granted a dropped role is immediately changed to reflect the absence of the dropped role privileges. All indirectly granted roles of the dropped role are also removed from affected security domains. Dropping a role automatically removes the role from all user default role lists.

Because the creation of objects is not dependent on the privileges received through a role, tables and other objects are not dropped when a role is dropped.

Syntax

DROP ROLE <ROLE_NAME>;

Example

DROP ROLE DEMO_ROLE;

It will drop the role called DEMO_ROLE that we defined earlier.

4.6 USER PROFILE

Profile is a set of limits on database resources. If you assign the profile to a user, then that user cannot exceed these limits. Use profiles to limit the database resources available to a user for a single call or a single session.

Prerequisites

- To create a profile, you must have the CREATE PROFILE system privilege.
- To specify resource limits for a user, you must:

- Enable resource limits dynamically with the ALTER SYSTEM statement or with the initialization parameter RESOURCE_LIMIT. This parameter does not apply to password resources. Password resources are always enabled.
- Create a profile that defines the limits using the CREATE PROFILE statement
- Assign the profile to the user using the CREATE USER or ALTER USER statement

Oracle Database enforces resource limits in the following ways:

- If a user exceeds the CONNECT_TIME or IDLE_TIME session resource limit, then the database rolls back the current transaction and ends the session. When the user process next issues a call, the database returns an error.

- If a user attempts to perform an operation that exceeds the limit for other session resources, then the database aborts the operation, rolls back the current statement, and immediately returns an error. The user can then commit or roll back the current transaction, and must then end the session.

- If a user attempts to perform an operation that exceeds the limit for a single call, then the database aborts the operation, rolls back the current statement, and returns an error, leaving the current transaction intact.

4.6.1. CREATE PROFILE

Syntax

CREATE PROFILE <PROFILE_NAME> LIMIT [Resource Parameter | Password Parameter] ;

UNLIMITED

When specified with a resource parameter, UNLIMITED indicates that a user assigned this profile can use an unlimited amount of this resource. When specified with a password parameter, UNLIMITED indicates that no limit has been set for the parameter.

DEFAULT

Specify DEFAULT if you want to omit a limit for this resource in this profile. A user assigned this profile is subject to the limit for this resource specified in the DEFAULT profile. The DEFAULT profile initially defines unlimited resources. You can change those limits with the ALTER PROFILE statement.

Any user who is not explicitly assigned a profile is subject to the limits defined in the DEFAULT profile. Also, if the profile that is explicitly assigned to a user omits limits for some resources or specifies DEFAULT for some limits, then the user is subject to the limits on those resources defined by the DEFAULT profile.

RESOURCE_PARAMETERS

- **SESSIONS_PER_USER:** Specify the number of concurrent sessions to which you want to limit the user.
- **CPU_PER_SESSION:** Specify the CPU time limit for a session, expressed in hundredth of seconds.
- **CPU_PER_CALL:** Specify the CPU time limit for a call (a parse, execute, or fetch), expressed in hundredths of seconds.
- **CONNECT_TIME:** Specify the total elapsed time limit for a session, expressed in minutes.
- **IDLE_TIME:** Specify the permitted periods of continuous inactive time during a session, expressed in minutes. Long-running queries and other operations are not subject to this limit.

- **LOGICAL_READS_PER_SESSION:** Specify the permitted number of data blocks read in a session, including blocks read from memory and disk.
- **LOGICAL_READS_PER_CALL:** Specify the permitted number of data blocks read for a call to process a SQL statement (a parse, execute, or fetch).
- **PRIVATE_SGA:** Specify the amount of private space a session can allocate in the shared pool of the system global area (SGA). Please refer to size clause for information on that clause.

PASSWORD_PARAMETERS

Use the following clauses to set password parameters. Parameters that set lengths of time are interpreted in number of days. For testing purposes you can specify minutes (n/1440) or even seconds (n/86400).

- **FAILED_LOGIN_ATTEMPTS:** Specify the number of failed attempts to log in to the user account before the account is locked.
- **PASSWORD_LIFE_TIME:** Specify the number of days the same password can be used for authentication. If you also set a value for PASSWORD_GRACE_TIME, the password expires if it is not changed within the grace period, and further connections are rejected. If you do not set a value for PASSWORD_GRACE_TIME, its default of UNLIMITED will cause the database to issue a warning but let the user continue to connect indefinitely.
- **PASSWORD_REUSE_TIME and PASSWORD_REUSE_MAX:** These two parameters must be set in conjunction with each other. PASSWORD_REUSE_TIME specifies the number of days before which a password cannot be reused. PASSWORD_REUSE_MAX specifies the number of password changes required before the current password can be reused. For these parameter to have any effect, you must specify an integer for both of them.
    - If you specify an integer for both of these parameters, then the user cannot reuse a password until the password has been changed the password the number of times specified for PASSWORD_REUSE_MAX during the number of days specified for PASSWORD_REUSE_TIME.

    - If you specify an integer for either of these parameters and specify UNLIMITED for the other, then the user can never reuse a password.
    - If you specify DEFAULT for either parameter, then Oracle Database uses the value defined in the DEFAULT profile. By default, all parameters are set to UNLIMITED in the DEFAULT profile. If you have not changed the default setting of UNLIMITED in the DEFAULT profile, then the database treats the value for that parameter as UNLIMITED.
    - If you set both of these parameters to UNLIMITED, then the database ignores both of them.
- **PASSWORD_LOCK_TIME:** Specify the number of days an account will be locked after the specified number of consecutive failed login attempts.
- **PASSWORD_GRACE_TIME:** Specify the number of days after the grace period begins during which a warning is issued and login is allowed. If the password is not changed during the grace period, the password expires.
- **PASSWORD_VERIFY_FUNCTION:** The PASSWORD_VERIFY_FUNCTION clause lets a PL/SQL password complexity verification script be passed as an argument to the CREATEPROFILE statement.

Examples

The following statement creates the profile named NEW_USER_PROFILE:

CREATE PROFILE NEW_USER_PROFILE LIMIT

PASSWORD_REUSE_MAX 10

PASSWORD_REUSE_TIME 30;

- **Check Your Progress**

21. How can any user Grant/Revoke a granted role to/from other users?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

22. How can user receive a Privileges?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

23. Explain Set Role Statement of SQL.

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

24. What is User Profile?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

4.7 LET US SUM UP

In this chapter, we have learnt about Role and Privileges. We have also concluded the system and object privileges. We have also explored different operation of User Role like Create, Grant and Revoke Role and Drop. We have come to know how can we set limits on resources for any user using profiles.

4.8 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

5. Any user Granted a role with ADMIN OPTION can Grant/Revoke that role to/from any other users.
6. A user can receive Privileges in two different ways.
    a. Grant Privileges to Users explicitly
    b. Grant Privileges to a Role and then Grant that Role to one or more users.

7. Set Role Statement is used to Enable or Disable a role for the current session.

8. User Profile is a set of limits on database resources and user cannot exceed these limits.

4.9 ASSIGNMENTS

1. Explain Privileges. Also describe difference between System Privileges and Object Privileges.
2. What is User Role? Describe with all options.
3. Explain User Profile in detail with all parameters.

4.10 Further Reading

1. SQL/PLSQL,TheProgrammingLanguageofORACLE,BPBPublicationbyIvan.
2. Introduction to Database Systems, 4th Edition, C. J. Date, Narose Publishing.

Block-4

Introduction to PL/SQL

Unit 1: Introduction to PL/SQL

1

Unit Structure

1.1. Learning Objectives

1.2. Introduction

1.3. PL/SQL Environment

1.4. Advantages of PL/SQL

1.5. Fundamentals of PL/SQL

1.6. Data types and Variables

1.7. Let Us Sum Up

1.8. Check Your Progress: Possible Answers

1.9. Assignments

1.10. Further Reading

1.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this unit is to make the students,

- To learn, understand basics of PL/SQL and its Block structure
- To learn, declare and initialize identifiers in PL/SQL block
- To learn, understand and access local and global variables

Outcome:

At the end of this unit,

- Students will be able to declare, initialize and access local and global variables
- Students will be able to write a PL/SQL block and execute it
- Students will be able to print the message or value from the PL/SQL block

1.2 INTRODUCTION

PL/SQL is Oracle's procedural language extension to SQL, a relational database language. PL/SQL thoroughly integrates modern software engineering features such as data encapsulation, information hiding, overloading, exception handling. We don't have a separate place or prompt for executing our PL/SQL programs. PL/SQL technology is like an engine that executes PL/SQL blocks and subprograms. Due to the strong integration of SQL and PL/SQL, PL/SQL is very effective in data manipulation.

SQL* Plus is an interactive and batch query tool that will be installed with every Oracle installation. We can found it at Start -> Programs -> Oracle-OraHomeName -> Application Development -> SQL Plus. It has also a command line user interface, Windows GUI, and web-based user interface. It allows the user to connect to the database and execute PL/SQL commands.

1.3 PL/SQL ENVIRONMENT

With PL/SQL, we can use SQL statements to manipulate ORACLE data and flow of control statements to process the data. Moreover, we can also declare constants, variables, define subprograms (procedures and functions) and handle runtime errors.

Thus, PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

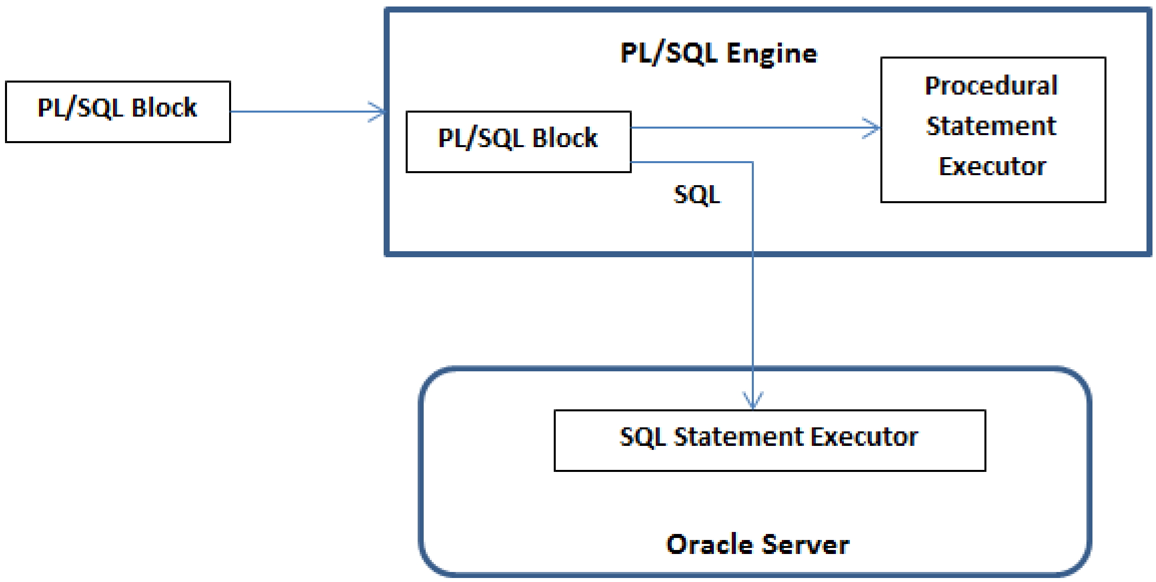


Figure 1 PL/SQL Environment

PL/SQL engine executes procedural statements and sends SQL part of statements to SQL statement processor in the Oracle server. PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

1.3.1 PL/SQL BLOCK STRUCTURE

PL/SQL is a block-structured language. i.e. Programs of PL/SQL contain logical blocks.

Declaration-Optional (Variable, Cursors, User Defined Exceptions)
Begin- Mandatory (SQL Statements, PL/SQL Statements)
Exception-Optional Error Handling code when error occurs
End; -Mandatory

Figure 2 PL/SQL Block Structure

As shown in the Figure 2 a PL/SQL block has three parts;

1. Declaration: Necessary variables are declared in this section. It is optional. This is an optional section of the code block. It contains the name of the local objects that will be used in the code block. These include variables, cursor definitions, and exceptions. This section begins with the keyword Declare.

2. Begin: This section contains executable statements of SQL and PL/SQL. This is the only mandatory section. It contains the statements that will be executed. These consist of SQL statements, DML statements, procedures (PL/SQL code blocks), functions (PL/SQL code blocks that return a value), and built-in subprograms. This section starts with the keyword Begin.

3. Exception: Any error occurred while executing the statements in begin part can be handled in this part. This is an optional section. It is used to "handle" any errors that occur during the execution of the statements and commands in the executable section. This section begins with the keyword Exception.

The code block is terminated by the End keyword. This is the only keyword within the construct that is followed by a semi-colon (;). The only required section is the executable section. This means the code block must have the Begin and End keywords. The code block is executed by the slash (/) symbol.

13.2.2 PL/SQL Block Types

There are three PL/SQL Block types as shown in figure 3.

Anonymous Block	Procedure Block	Function Block

[Declare]
Begin
---Statements
[Exception]
End;
	Procedure name
	is
	Begin
	---Statements
	[Exception]
	End;
		Function name
		Return datatype
		is
		Begin
		---Statements
		Return value;
		[Exception]
		End;

Figure 3 Pl/SQL Block types

PL/SQL is a block-structured language. The named blocks are called subprograms and unnamed blocks are called anonymous blocks. Subprograms can be referred as either functions or procedures. The difference between functions and procedures is that a function can be used in an expression and it returns a value to that expression. A procedure is invoked as a standalone statement and passes values to the calling program only through parameters. Subprograms can be nested within one another and can be grouped in larger units called packages. The basic units (procedures, functions, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. Typically, each logical block corresponds to a problem or sub-problem to be solved. Anonymous block don't have the name.

1.4 ADVANTAGES OF PL/SQL

There are various advantages of using PL/SQL. They are,

1. It is a portable and easy language.
2. We can declare identifiers.
3. We can program with procedural language control structures.
4. It can handle errors and prevents program from abnormal termination using the exception handling mechanism.
5. It modularizes program development through various PL/SQL blocks such as Procedure and functions.
6. It integrates with Oracle server and shared library.
7. It improves performance through better communication with underlying DBMSs.

1.5 FUNDAMENTALS OF PL/SQL

Lexical Units

PL/SQL is not case-sensitive language, so lower-case letters are equivalent to corresponding upper-case letters except within string and character literals. A line of PL/SQL text contains groups of characters known as lexical units, which can be classified as follows:

I. Delimiters (Simple and Compound Symbols)

A delimiter is a simple or compound symbol that has a special meaning to PL/SQL. For example, we can use delimiters to represent arithmetic operations such as addition and subtraction.

II. Identifiers (include Reserved Words)

We can use identifiers to name PL/SQL program objects and units, which include constants, variables, exceptions, cursors, subprograms and packages. Some identifiers called Reserved Words, have a special syntactic meaning to PL/SQL and so cannot be redefined. For flexibility, PL/SQL lets us to enclose identifiers within double quotes. Quoted identifiers are seldom needed, but occasionally they can be useful.

III. Literals

A literal is an explicit numeric, character, string, or Boolean value not represented by an identifier. Two kinds of numeric literals can be used in arithmetic expressions: integers and reals.

•String literal is a sequence of zero or more characters enclosed by single quotes. All string literals except the null string (`') belong to type CHAR. PL/SQL is case-sensitive within string literals.

•Boolean literals are the predefined values TRUE and FALSE and the non-value NULL (which stands for a missing, unknown, or inapplicable value). Boolean literals are not strings.

IV. Comments

The PL/SQL compiler ignores comments. Adding comments to our program enhances readability and guides the user in understanding the code. PL/SQL supports two types of comment styles, single-line and multiline.

• Single-line comments begin with a double hyphen (--) anywhere on a line and extend to the end of the line.

• Multiline comments begin with a slash asterisk (/*), end with an asterisk-slash (*/), and can span multiple lines. We cannot nest comments.

Example: In this code, we are going to print 'Welcome to GVP' and we are also going to check how the commented lines behave in the code.

```
BEGIN
 --This is a single line comment
dbms output.put line ('Welcome to GVP');
/*Multi line comments starts
Multi line comment ends */
END;
/
```

1.6 DATATYPES AND VARIABLES

Every constant and variable has a datatype, which specifies a storage format, constraints and valid range of values.

PL/SQL provides a variety of predefined scalar and composite datatypes. A scalar type has no internal components. A composite type has internal components that can be manipulated individually. PL/SQL mostly used datatypes are discussed below.

• NUMBER

We use the NUMBER datatype to store fixed or floating point numbers of virtually any size. We can specify precision, which is the total number of digits and scale, which determines where rounding occurs.

NUMBER[(precision, scale)]

We cannot use constants or variables to specify precision and scale; we must use an integer literals.

• CHAR

We use the CHAR datatype to store fixed-length character data. The CHAR datatype takes an optional parameter that lets us to specify a maximum length up to 32767 bytes.

CHAR[(maximum_length)]

We cannot use a constant or variable to specify the maximum length; we must use an integer literal. If we do not specify the maximum length, it defaults to 1.

• VARCHAR2

We use the VARCHAR2 datatype to store variable-length character data. The VARCHAR2 datatype takes a required parameter that lets us to specify a maximum length up to 32767 bytes.

VARCHAR2(maximum_length)

We cannot use a constant or variable to specify the maximum length; we must use an integer literal.

• BOOLEAN

We use the BOOLEAN datatype to store the values TRUE and FALSE and the non-value NULL. NULL stands for a missing, unknown, or inapplicable value. The BOOLEAN datatype takes no parameters.

• DATE

We use the DATE datatype to store fixed-length date values. The DATE datatype takes no parameters. Valid dates for DATE variables include January 1, 4712 BC to December 31, 4712 AD. When stored in the database column, date values will include the time of day in seconds since midnight. The default date portion is the first day of the current month and the default time portion is the midnight.

Defining Variables

Variables are defined in the declaration section of the program. The syntax is:

- Variable_name datatype(precision);

The d efinition m ust e nd w ith a s emi-colon. T he def inition s tatement b egins w ith t he variable name and contains the data type. A value may also be assigned to the variable during the definition statement. The variable may also be constrained.

Variables are used to store results. Forward references are not allowed. So we have to first declare the variable and then use it. Variables can have any SQL datatype, such as CHAR, D ATE, NUMBER et c or any P L/SQL datatype l ike B OOLEAN, BINARY_INTEGER etc.

We have to initialize variables designated as NOT NULL and CONSTANT. We have to initialize i dentifiers b y us ing t he as signment op erator (:=) or t he D EFAULT r eserved word.

Declaring Variables

Variables are declared in DECLARE section of PL/SQL.

```
DECLARE
Stu_No number (3);
Stu_Name varchar2 (15);
_ _ _
BEGIN
```

Variable Initialization

Variables and constants are initialized every time a block or subprogram is entered. By default, variables are initialized to NULL. So, unless you explicitly initialize a variable, its value i s undef ined. S calar v ariable dec laration an d i nitialization ex amples ar e as follows.

```
var_job VARCHAR2(9);
var _count BINARY_INTEGER := 0;
var _total_sal NUMBER(9,2) := 0;
var _orderdate DATE := SYSDATE + 3;
var _tax_rate CONSTANT NUMBER(3,2) := 8.25;
var _valid BOOLEAN NOT NULL := TRUE;
```

Constraints Definitions

Constraints can be placed on the variables defined in the code block. A constraint is a condition that is placed on the variable. Two common constraints are:

• **Constant:** This constraint will cause Oracle to ensure the value is not changed after a value is initially assigned to the variable. If a statement tries to change the variable value, an error will occur. The following is the example of constrained variable definitions:

PI constant number(9,8) := 3.14159265;

• **Not Null:** This constraint will cause Oracle to ensure the variable always contains a value. If a statement attempts to assign a null value to the variable, an error will occur. The following is the example of constrained variable definitions:

Date_of_Birth not null date := '26-March-2019';

Declaration and usage of variables:

Here we are going to print the 'Welcome to BAOU, Ahmedabad' using the variables and execute it.

```
Set Serveroutput on;
DECLARE
msg VARCHAR2(50);
BEGIN
msg:= 'Welcome to BAOU,Ahmedabad';
dbms_output.put_line (msg);
END:
/
```

Output:

Welcome to BAOU,Ahmedabad

SET SERVEROUTPUT ON

It is a command used to access results from Oracle Server. A PL/SQL program always followed by a slash ("/") on a line by itself. It sends the information to the compiler that the end of the block is reached. Without '/', the compiler will not consider the block is completed, and it will not execute it. DBMS_OUTPUT is a package and PUT_LINE is a function in it.

Scope of Variables

A variable in PL/SQL block is as local to that block and global to all its Sub-blocks. If we redeclare an identifier in a sub-block, we cannot reference the global identifier except we use a qualified name.

Example:

In the given example declaration two variables named num1 and num2 are in the outer block (i.e. Global variable) and third variable named num_sum declared into the inner block (i.e. local variable). Variable 'num_sum' is declared in inner block so can't access in the outer block. But no1 and no2 can be accessed anywhere in the block.

```
DECLARE
   no1 number := 25;
   no2 number := 15;
BEGIN
   DECLARE
num_sum number;
   BEGIN
num_sum := no1 + no2;
      dbms_output.put_line('Sum is: ' || num_sum);
   END;
END;
/
```

Output:

Sum is: 40

We can use OUTER keyword to access outer block variable inside the inner block. It is called global qualifier name space.

Example:

```
DECLARE
   no number := 25;
BEGIN
   DECLARE
      no number := 15;
   BEGIN
      IF no > OUTER.no THEN
        DBMS_OUTPUT.PUT_LINE('Inner variable is greater than outer variable');
      ELSE
        DBMS_OUTPUT.PUT_LINE('Inner variable is smaller than outer variable');
      END IF;
   END;
END;
/
```

Output:

Inner variable is smaller than outer variable

- **Check Your Progress**

1) What is the use of Dbms_output.put_line()?

…………………………………………………………………………………………………..
.
…………………………………………………………………………………………………..
.
…………………………………………………………………………………………………..
. ………………………………………………………………………………………
…………..

2) How do we get input from user in PL/SQL?

…………………………………………………………………………………………………..
.
…………………………………………………………………………………………………..

. ……………………………… ……………………………………………………………...

………………………………………………………………………………… ………….

3) While doing comparisons which rules to be applied to NULLs?

……………………………………………………………………………………………..

.

……………………………………………………………………………………………..

.

……………………………………………………………………………………………..

. ………………………………………………………………………………………

…………..

4) Write a PL/SQL program to add two numbers?

……………………………………………………………………………………………..

.

……………………………………………………………………………………………..

.

……………………………………………………………………………………………..

. ………………………………………………………………………………………

…………..

5) The PL/SQL engine executes the procedural commands and passes the SQL commands to the Oracle server to process. State **True** or **False.**

……………………………………………………………………………………………..

.

6) Explain types of PL/SQL blocks.

……………………………………………………………………………………………..

.

……………………………………………………………………………………………..

.

……………………………………………………………………………………………..

..………………………………………………………………………………………

1.7LET US SUM UP

In this unit, we have discussed about PL/SQL block, its benefit along with the use of SQL* Plus tool. We have also discussed about how to write the simple PL/SQL program and how to declare and use a variable in them. We have also used one package DBMS_OUTPUT to print the message.

1.8CHECK YOUR PROGRESS : POSSIBLE ANSWERS

- **Check Your Progress**

1. Dbms_output.put_line() statement takes a parameter which can be printed on to the console screen. When we start the SQL Command Prompt or Terminal, first we have to type:

   Set serveroutput on;

   This statement activates the working of print statement on the console screen.
2. We can get input from the user using the '&' sign. For example, to get input in to variable num,

   num:=#

   This statement will assign the value that the user enters for the variable.
3. While Comparison we need to keep in mind that,

   I. NULL will never be TRUE or FALSE

   II. NULL cannot be equal or unequal to other values

   III. When a value in an expression is NULL, then the expression itself evaluates to NULL except for concatenation operator (||)
4. Declare

```
        no1 integer;
        no2 integer;
        sum integer;
Begin
        no1:=& no1;
        no2:=& no2;
        sum:= no1 + no2;
```

```
        dbms_output.put_line(sum);
End;
/
```

5. True

6. PL/SQL blocks are of two types:

1. Anonymous blocks: A PL/SQL blocks without header are known as anonymous blocks.

These blocks do not form the body of a procedure, function or triggers.

Example:

```
DECLARE
        digit NUMBER(2);
        sqr NUMBER(3);
BEGIN
        digit:= &Number1;
        sqr:= digit  * digit;
        DBMS_OUTPUT.PUT_LINE('Square:' || sqr);
END;
```

2. Named blocks: PL/SQL blocks with header or labels are known as Named blocks. Named blocks may either be subprograms (procedures, functions, packages) or Triggers.

Example:

```
FUNCTION squar (digit IN NUMBER)
        RETURN NUMBER is sqr NUMBER(2);
BEGIN
        sqr:= digit * digit;
        RETURN sqr;
END;
```

1.9ASSIGNMENT

1. Define PL/SQL.

2. Discuss PL/SQL environment and block structure.

3. What is local and global variable access in PL/SQL block?

4. Discuss various advantages of PL/SQL.

5. Write a PLSQL code to check whether a number is prime or not.

1.10 FURTHER READING

1. SQL/PLSQL,TheProgrammingLanguageofORACLE,BPBPublicationbyIvan.

2. Introduction to Database Systems, 4th Edition, C. J. Date, Narose Publishing.

3. https://way2tutorial.com/plsql/

4. https://www.guru99.com/pl-sql-first-program-helloworld.html

Unit 2: Cursor

2

Unit Structure

2.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this unit is to make the students,

• To learn and understand Cursor and its execution cycle

• To define, declare and initialize Cursor to access data

• To learn and understand different types of Cursor

• To learn accessing Cursor through for loop

Outcome:

At the end of this unit,

• Students will be able to declare, initialize and access Cursor

• Students will be able to declare Cursor and write a PL/SQL block to access Cursor data

• Students will be able to write implicit, explicit and parameterized Cursor

14.2 INTRODUCTION

A cursor is a pointer to an area of memory, called a context area. The context area is allocated by oracle in order to process a SQL statement. The cursor allows PL/SQL to control what happens to the context area when a statement is processed. It can be used by user to process the output of a select statement that returns more than one row.

Oracle uses a work area to execute SQL commands and store processing information. PL/SQL allows us to access this area through a name using a Cursor. For the execution of every SQL statement certain area in memory is allocated. This private SQL area is called context area or Cursor. A cursor works as a handle or pointer into the context area.

When we declare a cursor, we get a pointer variable, which initially doesn't point anywhere. When the cursor is opened, memory is allocated and the cursor structure is created. The cursor variable will now points the cursor. When the cursor is closed the memory allocated for the cursor is released. Cursors allow the programmer to retrieve data from a table and perform actions on that data one row at a time.

2.3 CURSOR EXECUTION CYCLE

The important steps in the cursor execution cycle are OPEN, FETCH and CLOSE. A cursor execution cycle refers to the stages which a cursor follows to process and execute the query. The phases of cursor execution cycle are listed below:

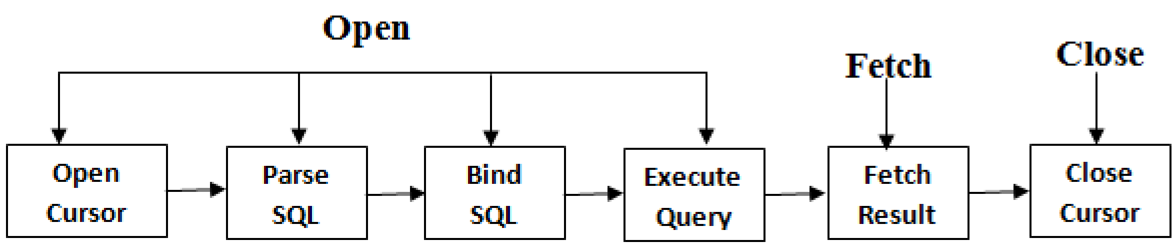


Figure 1: Cursor Execution Cycle

The activity carried out by the server in the key phases is:

1. OPEN Phase

In this phase,PGA memory is allocated for cursor processing, SELECT statement is parsed, Variable binding takes place, SELECT Query executes and finally pointer moves to the first record.

2. FETCH Phase

In this phase, the record to which the record pointer points, is retrieved from the result set. The record pointer will move only in the forward direction. The FETCH phase lives until the last record is reached.

3. CLOSE Phase

After the last record of the result set is reached, cursor is closed and allocated memory will be garbage collected and returned back to SGA. If an open cursor is not closed, oracle automatically closes it after the execution of its parent block.

2.4 Types of Cursor

There are two types of cursors.

- Implicit cursor
- Explicit cursor

2.4.1 IMPLICIT CURSORS

PL/SQL declares an implicit cursor for every DML command, queries it, which will return a single row. The name of the implicit cursor is SQL. We can directly use this cursor without any declaration.

For SQL queries which returns single row, PL/SQL declares implicit cursors. Implicit cursors are simple SELECT statements and are written in the BEGIN block (executable phase) of the PL/SQL. Implicit cursors retrieve exactly one row. The most commonly raised exceptions are NO_DATA_FOUND or TOO_MANY_ROWS.

For Example:

- Select sname, ssalary into sna, ssa from salesman where sno = 542;

Note: sname and ssalary are columns of the table salesman and sna and ssa are the variables

used to store sname and ssalary fetched by the query.

Oracle implicitly opens a cursor to process each SQL statement not associated with an explicitly declared cursor. We can refer to this cursor using the name SQL.

We cannot use the OPEN, FETCH, and CLOSE statements with SQL cursor. But, we can use cursor attributes to get information about the most recently executed SQL statement.

The following code shows how to use implicit cursor to know whether the most recent UPDATE has updated any rows or not.

```
DECLARE
        BEGIN
                update . . .
                if SQL%NOTFOUND then
                        statements;
                end if;
END;
```

NOTFOUND is an attribute of implicit cursor that will returns true if previous UPDATE command has not affected any row.

- **Implicit Cursor Attributes**

Cursor attributes do not have the similar meaning for both explicit and implicit cursors. The following are the attributes of implicit cursor.

1. NOTFOUND: It returns true, if previous DML operation didn't affect any row.
2. FOUND: It returns true, if previous DML operation affected any row.
3. ROWCOUNT: It returns number of rows affected by the most recent DML operation.

The following code shows how to use ROWCOUNT attribute with implicit cursor to know how many rows were updated with most recent UPDATE command.

```
BEGIN
        update salesman set scity = “Ahmedabad” where ssalary > 45;
        /* if more than 3 rows are effected then rollback updation */
        if SQL%ROWCOUNT > 3 then
                rollback;
        else
                commit;
        end if;
END;
```

2.4.2. EXPLICIT CURSOR

PL/SQL's implicit cursor can handle only single-row queries. But, if you need to select more than one row using select then you have to use explicit cursor. The set of rows fetched by a query is called active set. Select command in PL/SQL block will retrieve only one row. If select command retrieves no row then NO_DATA_FOUND exception will be raised. If select retrieves more than one row then TOO_MANY_ROWS exception occurs.

A select command will succeed only when it retrieves a single row. Select command copies the values of columns that it retrieved into variables. If multiple rows are retrieved then multiple values for each column will be copied to a single variable and that will create the problem.

Example:

```
DECLARE
        ssid varchar2(5);
        snam varchar2(5);
        sdpt varchar2(5);
    BEGIN
        select scode, sname, sdept into ssid, snam, sdpt
        from salesman where ssalary > 45;
END;
```

Select command in the above code will raise TOO_MANY_ROWS exception if more than one salesman is having salary more than 45.

An explicit cursor is the solution to the problem. A cursor can store a collection of records retrieved by a query. Then it allows us to fetch one record from cursor at a time and thereby enabling to process all the records in the cursor.

- **Handling Explicit Cursor**

Explicit cursor is a name used to refer to an area where you can place multiple rows retrieved by select. We must use an explicit cursor whenever we have to use a multi-row query in PL/SQL.

The following are the steps required to create and use an explicit cursor:

1. Declare the cursor in Declare section
2. Open the cursor using open statement in Executable part
3. Fetch one row at a time using fetch statement.
4. Close the cursor after all the records in the cursor are fetched and processed by using close.

Processing multiple rows is same as file handling. In file processing we need to open the file, process records and then close the file. Similarly user-defined explicit cursor needs to be opened, fetch and read the rows, after which it is closed. Like how file pointer marks current position in file processing, cursor marks the current position in the active set.

- **Declaring a Cursor**

A c ursor i s dec lared in D eclare s ection using c ursor s tatement. At t he t ime of declaration the
name of the cursor and the associated select statement are mentioned.

Syntax:

```
CURSOR cursor_name [(parameter[, parameter]...)]
IS select_statement
[FOR UPDATE [OF column,column, . . . ];
```

The following code shows how to declare a cursor.

```
DECLARE
            cursor sales_data is
            select scode, sname, sdept
            from salesman;
      BEGIN
      …………..
END;
```

sales_data is the name of the cursor, which will be populated with the rows retrieved by the
given select at the time of opening the cursor.

- **Opening a Cursor**

OPEN statement is used to execute the select command associated with the cursor and place
the rows retrieved by the query into cursor.

OPEN cursor_name [(input_arguments)];

Cursor_name is the name of the cursor that is to be opened.

Input_arguments are the values to be passed to the parameters of the cursor.

The following statement opens the cursor sales_data and places the rows retrieved by the query into the cursor.

```
DECLARE
        cursor sales_data is
        select scode, sname, sdept
        from salesman;
    BEGIN
        open sales_data;
END;
```

- **Fetching Rows**

Once cursor is opened using open statement, cursor has a set of rows, which can be fetched using fetch statement. Fetch statement takes the data of the current row in the cursor and copies the values of the columns into variables given after INTO keyword.

FETCH cursor_name INTO variable-1, variable-2, . . .;

For each column in the cursor there should be a corresponding variable in FETCH statement. We also need to make sure that the data types of variables and corresponding columns are matching.

The following code demonstrates how to fetch and copy data from current row of the cursor to variables given after INTO keyword.

```
DECLARE
        Cursor sales_data is
        select scode, sname, sdept
        from salesman;
        v_scode salesman.scode%type;
        v_sname salesman.sname%type;
        v_dept salesman.sdept%type;
    BEGIN
        open sales_data;
        loop
            fetch sales_data into v_scode, v_sname, v_dept;
            . . .
```

```
        end loop;
END;
```

FETCH statement is used inside the loop to repeatedly fetch rows from the cursor. The process of fetching will stop when all the rows of the cursor are fetched (reached end of cursor). T he f ollowing c ode s hows h ow t o exit c ursor w hen c ursor is c ompletely processed.

```
Loop
        fetch sales_data into v_scode, v_sname, v_sdept;
        exit when sales_data%notfound;
end loop;
```

NOTFOUND attribute of the c ursor r eturns T RUE when previous F ETCH do esn't successfully
read a row from cursor.

- **Closing a Cursor**

Close statement is used to close cursor after the cursor is processed. Closing a cursor will release the resources associated with cursor.

    CLOSE cursor_name;

The following code closes sales_data cursor:

```
DECLARE
    BEGIN
            open ..
            loop
            ...
            end loop;
        close sales_data;
END;
```

- **Explicit Cursor Attributes**

Cursor attributes allow user to retrieve information regarding cursor. For example, we can get the number of rows fetched so far from a cursor using ROWCOUNT attribute. We can also determine whether a row is fetched or not using FOUND attribute.

The following syntax is used to access cursor attributes:

cursor_name%Attribute

Every cursor defined by the user has 4 attributes. When appended to the cursor name these attributes allows the user to access important information about the execution of a multirow query.

The attributes are:

1. %NOTFOUND: It is a Boolean attribute, which returns true, if the last fetch is failed. i.e. when there are no rows left in the cursor to fetch.
2. %FOUND: Boolean variable, which returns true if the last fetch is succeeded.
3. %ROWCOUNT: It's a numeric attribute, which returns number of rows fetched by the cursor so far.
4. %ISOPEN: A Boolean variable, which returns true if the cursor is opened otherwise returns false.

The following code shows cursor attributes with explicit cursors. Attribute NOTFOUND returns true if previous FETCH statement couldn't fetch any row.

```
LOOP
        fetch sales_data into s_scode, s_dept;
        /* exit loop if previous FETCH failed */
        exit when sales_data%NOTFOUND;
        /* process the record fetched */
END LOOP;
```

In the above code EXIT is executed when NOTFOUND attribute of cursor sales_data returns TRUE.

- **Using Cursor with LOOP**

LOOP can be used to access the cursor values as shown in the following code.

Example:

```
DECLARE
        Lname varchar2(10);
        Sal number(8,2);
        CURSOR C1 IS Select Last_Name, Salary from Employee;
BEGIN
        Open C1;
        dbms_output.put_line('Last_Name'||' '||'Salary');
        If C1%isopen then
                LOOP
                        Fetch C1 into Lname, Sal;
                        dbms_output.put_line(Lname||' '||Sal);
                END LOOP;
        END IF;
END;
/
```

Fetch is used twice in the below example using While Loop to make %FOUND available.

Example:

```
DECLARE
        Cursor C1 is
        SELECT ID, Last_Name, city FROM Employee;
        Num Employee.ID%type;
        Nam Employee.Last_Name%type;
        Town Employee.city%type;
Begin
Open C1;
 Fetch C1 into Num, Nam, Town;
while C1%found loop
```

```
        dbms_output.put_line('Row Number '||C1%rowcount || ' is: '|| Num||' '||Nam||'
'||Town);
        Fetch C1 into Num, Nam, Town;
End loop;
Close C1;
End;
/
```

The above code will display the cursor C1 records with Employee Id, Name and city.

2.5 CURSOR FOR LOOP

The cursor for Loop can be used to process multiple records. There are two benefits with cursor for Loop.

1. It implicitly declares a %ROWTYPE variable.
2. Cursor for loop itself opens a cursor, read records and then closes the cursor automatically. So, Open, Fetch and Close statements are not necessary in it. To process a cursor, we can use cursor FOR loop to automate the following steps.

- Opening cursor
- Fetching rows from the cursor
- Terminating loop when all rows in the cursor are fetched
- Closing cursor

The following is the syntax of cursor for loop. This for loop is specifically meant to process cursors.

```
FOR rowtype_variable IN cursor_name
        LOOP
                Statements;
END LOOP;
```

rowtype_variable is automatically declared by cursor for loop. It is of ROWTYPE of the cursor. It has columns of the cursor as fields. These fields can be accessed using rowtype_variable.fieldname.

Example:

```
DECLARE
        CURSOR C1 IS Select Last_Name, Salary from Employee;
BEGIN
        For EMP_REC in C1
        LOOP
                        dbms_output.put_line(EMP_REC.Last_name||’
                        ‘||EMP_REC.Salary);
        END LOOP;
END;
/
```

The above code will display the cursor C1 records with Employee Last Name and their salary. emp_rec is automatically created variable of %ROWTYPE. We have not used Open, Fetch and Close in the above example as cursor for loop does it automatically. Using Implicit for Loop the above example can be rewritten as shown below:

Example:

```
BEGIN
        For EMP_REC in (Select Last_Name, Salary from Employee)
        LOOP
                        dbms_output.put_line(EMP_REC.Last_name||’
                        ‘||EMP_REC.Salary);
        END LOOP;
END;
/
```

____

2.6 Parameterized Cursor

Parameterized Cursor passes the parameters into a cursor and uses them in the query. PL/SQL parameterized cursor define only datatype of parameter and doesn't require to define it's length. A cursor FOR loop automatically opens the cursor to which it refers, so our program doesn't require opening that cursor inside the loop.

Syntax: The syntax for a cursor with parameters in PL/SQL is:

```
CURSOR cursor_name (parameter_list)
IS
  SELECT_statement;
```

Example:

```
DECLARE
        Cursor C1(num number) is select * from Employee
        where ID = num;
        emp Employee%rowtype;
BEGIN
      If C1%Isopen Then
      Close C1;
      End If;
        -- Open C1(5);
        FOR emp IN C1(5) LOOP
        dbms_output.put_line('EMP_NUM:    '||emp.ID);
        dbms_output.put_line('First_Name:  '||emp.First_Name);
        dbms_output.put_line('Last_Name:  '||emp.Last_Name);
        dbms_output.put_line('EMP_Salary:'||emp.Salary);
        END Loop;
-- CLOSE C1;
END;
/
```

- **Check Your Progress**

1) What is a cursor? Why Cursor is required?

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

2) Write the PL/SQL Statements used in cursor processing.

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

3) Write the cursor attributes used in PL/SQL.

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

4) Check following code and tell what will happen after commit statement?

```
Cursor C1 is
      Select empno,
      ename from emp;
Begin
       open C1;
loop
       Fetch C1 into
eno. ename;
       Exit When
         C1 %notfound;-----
       commit;
end loop;
end;
```

5) What is the use of WHERE CURRENT OF clause in cursors?

……………………………………………………………………………………………..

……………………………………………………………………………………………..

…………………………………………………………………………………………..…

2.7SUMMARY

In this unit we have learnt that the major task of a cursor is to fetch data, one row at a time, from the result set. Cursors are used whenever the user wants to manipulate or update records in a singleton fashion or in a row by row manner, in a database table. The information stored in the Cursor is known as Active Data Set. Cursors are opened in predefined area of Oracle's DBMS in the main memory set, where the cursors are opened. We have also discussed cursor with for loop and parameter. Cursor plays an important role in accessing data one row at a time unlike sql commands.

2.8 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

- **Check Your Progress**

1. Cursor is a named private SQL area from where we can access information. Cursors needs to process rows individually for queries returning multiple rows.

2. DECLARE CURSOR cursor name, OPEN cursor name, FETCH cursor name INTO or Record types, CLOSE cursor name.

3. Cursor attributes are;

   I. %ISOPEN : It is used to check whether cursor is open or not.

   II. % ROWCOUNT : It returns the number of rows fetched / updated / deleted.

   III. % FOUND : It is used to check whether cursor has fetched any row. Returns true if rows are fetched.

   IV. % NOT FOUND : It is used to check whether cursor has fetched any row. Returns true if no rows are fetched.

These attributes are processed with SQL for Implicit Cursors and with Cursor name for Explicit Cursors.

4. In the above code the cursor is having query SELECT, so does not get closed even after Commit / Rollback.

If, the cursor is having query as SELECT .... FOR UPDATE then it gets closed after Commit / Rollback.

5. In cursor, WHERE CURRENT OF clause in an Update, Delete statement refers to the latest row retrieved from a cursor.

2.9 ASSIGNMENTS

1. Define Cursor. Explain Cursor Cycle.
2. Discuss the types of cursor with proper syntax.
3. How do we use While Loop and For Loop in Cursor? Discuss with example.
4. Explain parameterized Cursor with example.
5. Differentiate Cursor declared in a procedure and Cursor declared in a package specification.
6. What are PL/SQL cursor exceptions?

2. 10 FURTHER READING

1. Advanced PL/SQL Programming: The Definitive Reference by Boobal Ganesan
2. SQL/PLSQL,TheProgrammingLanguageofORACLE,BPBPublicationbyIvan.
3. Introduction to Database Systems, 4th Edition, C. J. Date, Narose Publishing.

Unit 3: Locking

3

Unit Structure

3.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this unit is to make the students,

- To learn and understand database lock
- To learn the benefits of locking any database objects
- To learn and understand different modes of locks
- To learn and understand different types of locks

Outcome:

At the end of this unit,

- Students will be able to define database lock
- Students will be able to lock table with different locking mode

3.2 INTRODUCTION

Oracle Database provides data concurrency, consistency and integrity among transactions through a locking mechanism. The locks are performed automatically and require no user interaction. It is directly associated with a session. Database Locks are mechanisms that prevent destructive interaction between transactions accessing the shared resource or objects. These resources can be tables, data rows, data blocks, cached items, connections and entire systems.

There are many types of locks that can occur such shared locks, exclusive locks, transaction locks, DML locks, and backup-recovery locks. Oracle database automatically obtains required locks when performing SQL transactions. For example, before a session is permitted to update data, the session must first lock the data. The lock empowers the session exclusive control over the data so that no other transaction can update the locked data until the lock is released.

3.3 Locking Strategy

The database maintains different types of locks based on the operation that hold the lock. Locks have direct impact on the interaction of read and write operation. The following rules summarize the locking behaviour of oracle database for reads and writes:

- A row is locked whenever modified by a write operation. When a transaction updates one row, the transaction acquires a lock for this row only. The contention can be minimized by locking table data at the row level.
- When one transaction is updating a row, then a row lock prevents a different transaction from updating the same row concurrently.
- A read operation never blocks a write operation. A reading of a row does not lock that row, a write operation can update this row. The only exception is a SELECT ... FOR UPDATE statement that will lock the row being read.
- A write operation never blocks a read operation. When a row is being changed by a write transaction, the database applies undo data to provide readers with a consistent view of the row data.

3.3.1. LOCK MODES

Following table describe various types of locking mode with their meaning.

Lock Mode	**Meaning**
EXCLUSIVE	It allows a SELECT query on the locked table, all other operations (i.e. Update, Delete etc.) are prohibited to other transactions.
SHARE	It allows concurrent queries, but updates are prohibited for all transactions.

Lock Mode	Meaning
ROW SHARE	It allows concurrent access to the table, but no other users can acquire an exclusive lock on the table.
ROW EXCLUSIVE	It is essentially the same as ROW SHARE but also prevents locking in SHARE mode.
SHARE ROW EXCLUSIVE	It locks the entire table; queries are allowed but no other transaction can acquire any lock on the table.

3.4 Types of Lock

Oracle server implicitly acquires a lock situation if a transaction is done on the same table in different sessions. This default locking technique is called implicit or automatic locking.

In Explicit Locking, a table or partition can be locked using the LOCK TABLE statement in one of the earlier specified modes. It is better to acquire an Explicit Locking rather than relying on the implicit locking done by default by the Oracle server.

Generally, the database uses two types of locks:

3.4.1 EXCLUSIVE LOCKS

In Exclusive locks only one lock can be obtained on an object such as a row or a table. This locking mode prevents the associated resource from being shared. A transaction acquires an exclusive lock when it updates data. The first transaction who had acquired a lock to resource exclusively is the only transaction that can modify the resource until the exclusive lock is released.

15.3.2. Shared locks

In Shared locks many share locks can be obtained on a single object. This locking mode allows the associated resource to be shared based on the operations involved.

Multiple users reading data can share the same data, acquiring share locks to prevent simultaneous access by a write transaction looking for an exclusive lock.

Oracle database does not allow a field level locking. It gives the Row level, Page level and Table level locking mechanism.

I. Row Level locking

In row-level locking, any specific row or rows in a table can be locked (unlocked rows w ill be av ailable f or upd ates or del etes). T he l ocked r ows c an be updat ed only by the process that initiated the locking.

II. Page Level locking

A page level locking is used when the Where clause evaluates to a set of data.

III. Table Level locking

In table-level locking, the whole table is locked against any kind of DML actions from

another transaction. Once a given transaction has locked a table, that transaction is the

only one that can change rows in the table.

3.5 LOCK TABLE

To lock any database table following syntax can be used.

Syntax:

- LOCK TABLE tables IN lock_mode MODE [WAIT [, integer] | NOWAIT];

Where,

- Tables is a A comma-delimited list of tables,
- lock_mode is a previously discussed any lock mode,
- WAIT s pecifies t hat t he da tabase w ill w ait f or a s pecific n umber of s econds as mentioned by integer to acquire a DML lock.
- NOWAIT indicates that the database should not wait for a lock to be released.

Example

Let's look at below code of how to use the LOCK TABLE statement.

For example:

- LOCK TABLE Student IN SHARE MODE NOWAIT;

This code will lock the Student table in SHARE MODE and not wait for a lock to be released.

- Lock table Student IN Exclusive Mode NOWAIT;

Above code will lock the Student table in EXCLUSIVE MODE and not wait for a lock to be released.

- **Check Your Progress**

1) What are LOCKS?

……………………………………………………………………………………………….

……………………………………………………………………………………………….

……………………………………………………………………………………………….

2) Write two important database goals of Locking.

……………………………………………………………………………………………….

……………………………………………………………………………………………….

……………………………………………………………………………………………….

3) Write different types of locks available in database.

……………………………………………………………………………………………….

……………………………………………………………………………………………..
.
……………………………………………………………………………………………..
.

4)What will happen if another session tries to update the locked data?

………………………………………………………………………………………………
………………………………………………………………………………………………
……......…………………………………………………………………………………….
.

3.6LET US SUM UP

Locking is a mechanism to ensure data consistency, concurrency and integrity while allowing maximum simultaneous access to objects. It is used to implement concurrency control when multiple users try to manipulate table data at the same time. By learning locking we can say that it helps in avoiding deadlock conditions and also avoids clashes in acquiring the database resources. Generally a user does not need to worry about locking, as RDBMS automatically selects the most appropriate lock for a particular transaction.

3.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

- **Check Your Progress**

1. Locks are techniques used to prevent destructive interaction between users accessing database objects. ORACLE uses locks to control concurrent access to data.
2. I. Consistency: It ensures that the data objects a user is reading or changing is not changed (by other users) until the user is finished with the data.
   II. Integrity: It ensures that the database's data object and structures reflect all changes made to them in the correct order.
3. a. Data Locks (DML)
   b. Dictionary Locks (DDL)
   c. Internal Locks and Latches

d. Distributed Locks

e. Parallel Cache Management Locks

4. Suppose database session A tries to update some data that is already locked by database
session B. Here, session A will remain in lock wait state, and session A will be stopped from making any progress with any SQL transaction that it's executing. We can say that session A will be blocked until session B releases the lock on that data.

3.8ASSIGNMENTS

1. Define Lock. Explain Locking benefits.
2. Discuss different types of locking with example.
3. Explain various modes of lock.

3.9FURTHER READING

1. Advanced PL/SQL Programming: The Definitive Reference by Boobal Ganesan
2. SQL/PLSQL,TheProgrammingLanguageofORACLE,BPBPublicationbyIvan.
3. Introduction to Database Systems, 4th Edition, C. J. Date, Narose Publishing.

Unit 4: Exception Handling 4

Unit Structure

4.1 LEARNING OBJECTIVES & OUTCOMES

The objective of this unit is to make the students,

• To learn and understand Exception

• To define and understand different types of Exception

• To learn and understand Exception handling

Outcome:

At the end of this unit,

• Students will be able to write exception handling block

• Students will be able to declare user defined exception

• Students will be able to use pre-defined exception for different types of errors

• Students will be able to write pragma exception

4.2INTRODUCTION

An Exception is an error situation or abnormal condition, which arises during program execution. When an error takes place exception is raised, normal execution is stopped and control transfers to exception handling block. Exception handlers are block of codes written to handle the exception. The exceptions can be system-defined or pre-defined and User-defined exception. When PL/SQL raises a predefined exception, the program is terminated by displaying error message. But if the program is supposed to handle exception raised by PL/SQL then we have to use Exception Handling part of the block.

Control is transferred to exception handling part whenever an exception occurs. After the exception handler completes execution, control is transferred to next statement in the enclosing block. If there is no enclosing block then control returns to Host (from where we ran the PL/SQL block).

Syntax of exception handling is:

```
WHEN exception-1 [or exception -2] ... THEN
statements;
[WHEN exception-3 [or exception-4] ... THEN
```

```
statements; ] ...
[WHEN OTHERS THEN
statements; ]
```

exception-1, exception-2 are exceptions that are to be handled. These exceptions are either pre-defined exceptions or user-defined exceptions. If an exception is raised but not handled by exception handling part then PL/SQL block is terminated by displaying an error message related to the exception.

The biggest advantage of exception handling is that it improves readability and reliability of the code. Errors from many statements of code can be handles with a single handler. Instead of checking for an error at every point we can just add an exception handler to handle the exception when raised.

4.3 USER-DEFINED EXCEPTIONS

A User-defined exception is an exception defined by the programmer. User-defined exceptions are declared in the declaration section with their type as exception. They must be raised explicitly using RAISE Command, while pre-defined exceptions are raised implicitly. RAISE statement can also be used to raise internal exceptions. We can map exception names with specific Oracle errors using the EXCEPTION_INIT Pragma. We can also assign a number and description to the exception using RAISE_APPLICATION_ERROR.

Declaring Exception:

```
DECLARE
        myexception EXCEPTION;
BEGIN
        Raising Exception:
BEGIN
         RAISE myexception;
Handling Exception:
BEGIN
          EXCEPTION
```

```
        WHEN myexception THEN
        Statements;
END;
```

Note:

- An Exception cannot be declared twice in the same block.
- Exceptions declared in a block are considered as local to that block and global to its sub-
blocks.
- An enclosing block cannot access Exceptions declared in its sub-block. While it is possible for a sub-block to refer its enclosing Exceptions.

The f ollowing example d emonstrates t he us e of User-defined E xception us ing Procedure**:**

```
Create or Replace Procedure Raise_Exception (Input NUMBER) IS
        Evenno_Exception EXCEPTION;
        Oddno_Exception EXCEPTION;
Begin
        IF MOD(Input, 2) = 1 THEN
          RAISE Oddno_Exception;
        ELSE
          RAISE Evenno_Exception;
        END IF;
       EXCEPTION
        WHEN Evenno_Exception THEN
          dbms_output.put_line(TO_CHAR(Input) || ' is Even Number ');
        WHEN Oddno_Exception THEN
          dbms_output.put_line(TO_CHAR(Input) || ' is Odd Number');
End Raise_Exception;
/
```

Now execute the procedure with following command and check out the output as shown below.

- exec Raise_Exception(5);

  5 is odd Number

4.3.1 RERAISING AN EXCEPTION

When we want an exception to be handled in the current block as well in its enclosing block then we need to use RAISE statement without an exception name. RAISE command can also be used to reraise an exception so that the current exception is propagated to outer block. Current exception will be raised again if a sub block executes RAISE statement without specifying exception name in exception handler. In the below example, the exception ZERO_DIVIDE is logged into a table before it is re-raised to the user or to the application.

Note: RAISE statement without exception name is valid only in exception handler.

```
DECLARE
 num NUMBER;
BEGIN
        num := 5/0;
        EXCEPTION
        WHEN zero_divide THEN
        INSERT INTO log_details VALUES (log_seq.nextval, SQLCODE ||' '||
sqlerrm);
        RAISE;
END;
/
```

4.3.2 RAISE APPLICATION ERROR

To display our own error messages we can use the built in RAISE_APPLICATION_ERROR. It will display the error message in the same way as Oracle errors. We should use a negative number between –20000 to –20999 for the error_number and the error message should not exceed 512 characters.

Syntax:

```
RAISE_APPLICATION_ERROR(<error_number>, <error_message>, <TRUE |
FALSE>);
          Where,
          error_number -20000 to -20999
          error_message Varchar2(2048)
          TRUE          add to error stack
          FALSE         replace error stack (the default)
```

Let’s try to understand with following example.

```
CREATE OR REPLACE PROCEDURE Raise_application_Exception (Input NUMBER)
IS
           evenno_exception EXCEPTION;
           oddno_exception  EXCEPTION;
BEGIN
            IF MOD(Input, 2) = 1 THEN
              RAISE oddno_exception;
            ELSE
              RAISE evenno_exception;
            END IF;
          EXCEPTION
            WHEN evenno_exception THEN
              RAISE_APPLICATION_ERROR(-20001, 'Even Number Entered');
            WHEN oddno_exception THEN
              RAISE_APPLICATION_ERROR(-20999, 'Odd Number Entered');
END Raise_application_Exception;
/
```

Execute the above procedure with following command and check the output. It will display error message with error number.

- Exec Raise_application_Exception(5);

4.4 Predefined (Named) Exceptions

Predefined exception is raised automatically whenever there is a violation of Oracle coding rules. PL/SQL has defined certain common errors and given names to these errors, which are called as predefined exceptions. Each exception has a corresponding Oracle error code. Predefined exceptions examples are those like ZERO_DIVIDE, which is raised automatically when we try to divide a number by zero. Other built-in exceptions are given below. We can handle unexpected Oracle errors using OTHERS handler. It can handle all raised exceptions that are not handled by any other handler. It must always be written as the last handler in exception block. Predefined exception handlers are declared globally in package Standard. We don't need to define them.

Structure of Error Handling:

```
CREATE OR REPLACE PROCEDURE <procedure_name> IS
BEGIN
  NULL;
EXCEPTION
  WHEN <named_exception> THEN
    -- handle identified exception
  WHEN <named_exception> THEN
    -- handle identified exception
  WHEN OTHERS THEN
    -- handle any exceptions not previously handled
END;
/
```

Example of ZERO_DIVIDE Exception:

```
Declare
   num    number := 50;
   div number := 0;
   result number;
```

```
begin
   result := num / div;
   dbms_output.put_line('result: '||result);
exception
   when zero_divide then
   dbms_output.put_line('DIVIDE by ZERO: '||sqlerrm);
end;
/
```

Example of NO_DATA_FOUND Exception:

The below program will show the name and address of a salesman as result whose ID is matches. But there is no salesman with ID 10 in our record, so the program raises the run-time exception NO_DATA_FOUND, which is captured in EXCEPTION block.

```
DECLARE
  s_id salesman.id%type := 10;
  s_name  salesman.name%type;
  s_addr salesman.address%type;
BEGIN
  SELECT  name, address INTO  s_name, s_addr
  FROM salesman
  WHERE id = s_id;
DBMS_OUTPUT.PUT_LINE ('Name: '||  s_name);
 DBMS_OUTPUT.PUT_LINE ('Address: ' || s_addr);
EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('No such Salesman exists!');
  WHEN others THEN
    dbms_output.put_line('There is problem'');
END;
/
```

The **DUP_VAL_ON_INDEX** exception is raised when a SQL statement tries to create a duplicate value in a column on which primary key or unique constraints are defined. Following example demonstrates the use of DUP_VAL_ON_INDEX exception.

```
BEGIN
    Insert into salesman (id) values(1);
    EXCEPTION
              When dup_val_on_index then
               dbms_output.put_line('Duplicate value on an index');
END;
 /
```

More than one Exception can be written in a single handler as shown below.

```
EXCEPTION
When NO_DATA_FOUND or TOO_MANY_ROWS then
Statements;
END;
```

Invalid Cursor Exception

Here we will try to check the exception associated with Cursor access. Let's examine the below example.

```
CREATE OR REPLACE PROCEDURE InvalidCursor_exception IS
 CURSOR CurExcp is
 SELECT *  FROM salesman;
 Cur_Record CurExcp%rowtype;
BEGIN
  LOOP
   -- note the cursor was not opened before the FETCH
   FETCH CurExcp INTO Cur_Record;
   EXIT WHEN CurExcp%notfound;
```

```
    NULL;
  END LOOP;
EXCEPTION
  WHEN INVALID_CURSOR THEN
    dbms_output.put_line('Invalid Cursor State exception Raised');
  WHEN OTHERS THEN
    dbms_output.put_line('Some Other Problem');
END InvalidCursor_exception;
/
```

Execute the above procedure and check the output.

The following table shows some important predefined exception with their meaning and error code.

Exception Name	**Error**	**Description**
CASE_NOT_FOUND	ORA-06592	None of the choices in the WHEN clauses of a CASE statement is selected and there is no ELSE clause.
CURSOR_ALREADY_OPEN	ORA-06511	Raised when tried to open a cursor that was already open
DUP_VAL_ON_INDEX	ORA-00001	Raised when an attempt to insert or update a record in violation of a primary key or unique constraint is made
INVALID_CURSOR	ORA-01001	Raised when the cursor is not open, or not valid in the context in which it is being called.
INVALID_NUMBER	ORA-01722	Raised when it isn’t a number
LOGIN_DENIED	ORA-01017	Invalid name and/or password for the instance.

NO_DATA_FOUND	ORA-01403	Raised when the SELECT statement returned no rows or referenced a deleted element in a nested table or referenced an initialized element in an Index-By table.
NOT_LOGGED_ON	ORA-01012	Raised when database connection lost.
PROGRAM_ERROR	ORA-06501	Raised when internal PL/SQL error.
ROWTYPE_MISMATCH	ORA-06504	Raised when the rowtype does not match the values being fetched or assigned to it.
STORAGE_ERROR	ORA-06500	Raised when a hardware problem either RAM or disk drive occurs.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Raised when reference to a nested table or varray index higher than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Raised when reference to a nested table or varray index outside the declared range (such as -1).
TIMEOUT_ON_RESOURCE	ORA-00051	Raised when the activity took too long and timed out.
TOO_MANY_ROWS	ORA-01422	Raised when the SQL INTO statement brought back more than one value or row (only one is allowed).
ZERO_DIVIDE	ORA-01476	Raised when an attempt is made to divide a number by zero.

4.5 SQLCODE AND SQLERRM

In W HEN O THERS par t of ex ception h andler, w e c an us e S QLCODE and S QLERRM functions t o r etrieve t he er ror num ber an d er ror m essage r espectively. T here i s no predefined exception for every oracle errors.

By using these two functions we can get the error code and error message of the most recently oc curred er ror. T he f ollowing ex ample d emonstrates how t o us e S QLCODE and S QLERRM f unctions. T o un derstand t his w e w ill c reate a t able nam ed s ubject as follows.

- Create t able s ubject(subcode v archar2(2) pr imary k ey no t nul l, s ubname varchar2(20));

After creating Table insert few records as shown below. Here we have to define subject code primary key and not null.

- Insert into subject values(‘A’,’Java’);
- Insert into subject values(‘B’,’DBMS’);
- Insert into subject values(‘C’,’RDBMS’);
- Insert into subject values(‘D’,’C++’);

Now write and execute following code and check the output.

Example:

```
        Declare
        newscode varchar2(5) := null;
        begin
        update subject set subcode = newscode where subcode = 'C';
        exception
        when dup_val_on_index then
        dbms_output.put_line('Duplicate subject code');
        when others then
        dbms_output.put_line(sqlerrm);
end;
/
```

If you run the above program, it will show cannot update ('SYSTEM','Subject','subcode') to null with error code ORA-01407.

The above output is generated when others part of exception handling block executes. SQLERRM returns the error message of the most recent error occurred. As we are trying to set SCODE, which is a not null column to NULL value, PL/SQL raises an exception. But as the error (-01407) is not associated with any predefined exception, WHEN OTHERS part of exception handling part is executed.

4.6 PRAGMA EXCEPTION

PRAGMA EXCEPTION_INIT allows user to map ORA- error and it can be raised in PL/SQL code. The SQL Error number passed in "EXCEPTION_INIT" is the same as error code except for "NO_DATA_FOUND" ORA-01403 which is 100.

Example:

```
Declare
no_rows_found exception;
pragma exception_init(no_rows_found, 100);
Begin
raise no_rows_found;
End;
/
```

Execute above code and check the output.

Example with too many rows:

```
Declare
too_many_rows exception;
Pragma exception_init(too_many_rows, -1422);
Begin
raise too_many_rows;
End;
/
```

Execute above code and check the output.

Whenever Oracle error -1407 occurs, NULL_VALUE_ERROR exception is raised by PL/SQL. The following example will illustrate important points related to associating an Oracle error with a user-defined exception.

Here we will consider the previously created Subject table and same update query for assigning null value to a not null column.

Example:

```
Declare
        null_value_error Exception;
        Pragma Exception_init(null_value_error, -1407);
        newscode varchar2(5) := null;
begin
        update subject set subcode = newscode where subcode = 'C';
        Exception
        When null_value_error Then
        dbms_output.put_line('User is trying to set null value to a not null column');
end;
/
```

Execute above code and check the output.

- **Check Your Progress**

1) What is an Exception? State the types of Exception.

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

2) What do you mean by PRAGMA keyword?

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

3) What is Raise_application_error?

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

4) What is the benefit of OTHERS exception handler?

……………………………………………………………………………………………….

………..………………………………………………………………………………………

………………………………………………………………………………………………

5) What is PRAGMA EXECPTION_INIT? Explain its use?

…………………………………………………………………………………………………

…………..……………………………………………………………………………………

………………………………………………………………………………………………

4.7 LET US SUM UP

A PL/SQL block is successful if it exits without raising any exceptions or raises an exception but the exception is handled in the block's exception handling part. Same way, A PL/SQL block is unsuccessful if it exits with an unhandled exception means the executable part raises an exception (either predefined or user-defined) and it is not handled in the block's exception handler. In this unit we have discussed the exception and exception handling mechanism using predefined and user defined exception. We have also discussed RAISE_APPLICATION_ERROR procedure to generate a user-defined error.

4.8CHECK YOUR PROGRESS:POSSIBLE ANSWERS

- **Check Your Progress**

1. Exception is an error and Exception handling is the error handling part of PL/SQL block. The types of Exception are Predefined and user_defined. Some of Predefined exceptions are:

- CURSOR_ALREADY_OPEN

- DUP_VAL_ON_INDEX
- NO_DATA_FOUND
- TOO_MANY_ROWS
- INVALID_CURSOR
- INVALID_NUMBER
- LOGON_DENIED
- NOT_LOGGED_ON
- PROGRAM-ERROR
- STORAGE_ERROR
- TIMEOUT_ON_RESOURCE
- VALUE_ERROR
- ZERO_DIVIDE
- OTHERS.

2. The PRAGMA keyword specifies that the statement is a compiler directive, which is not processed when the PL/SQL block is executed. It is a pseudo-code that tells the compiler to interpret all the occurrences of exception name within the block with the associated oracle server number.

3. Raise_application_error is a procedure of package DBMS_STANDARD. It allows issuing an user_defined error messages from stored sub-program or database trigger.

4. The OTHERS exception handler makes sure that no exception goes unhandled and the program terminates successfully.

5. The PRAGMA EXECPTION_INIT informs the complier to associate an exception with an oracle error to get an error message of a specific oracle error.

For example, PRAGMA EXCEPTION_INIT (exception name, oracle error number)

4.9ASSIGNMENT

1. What is Exception? How do we handle Exception in PL/SQL?
2. Explain User defined exception in PL/SQL.
3. Write a PL/SQL code to explain any four predefined exception.
4. Discuss PRAGMA Exception.
5. Discuss the SQLCODE and SQLERRM functions.

6. Is it possible for a PL/SQL block to process more than one exception at a time?

4.10FURTHER READING

1. Advanced PL/SQL Programming: The Definitive Reference by Boobal Ganesan
2. SQL/PLSQL,TheProgrammingLanguageofORACLE,BPBPublicationbyIvan.
3. Introduction to Database Systems, 4th Edition, C. J. Date, Narose Publishing.