

UNIT I INTRODUCTION TO DATABASE SYSTEM CONCEPTS AND ARCHITECTURE

CHAPTER 1 DATA BASES AND DATABASE USERS

1.1 Introduction:

Database: A database is a collection of related data. Data is a collection of raw facts that can be recorded. Database has the following properties

1. It represents some aspects of real world.
2. It is a logically coherent collection of data with some inherent meaning
3. It is designed, built and populated with data for a specific purpose
4. It has an intended group of users and some preconceived applications in which these users are interested

E.g. An address book containing names, addresses and phone numbers of people in a software like MS-Excel or MS-Access

DBMS- Database Management System: It is a collection of programs that enables users to create and maintain databases. It is a general purpose software system that facilitates the processes of defining, constructing, manipulating and sharing the database amongst various users and applications.

1.1.1 Functions of DBMS:

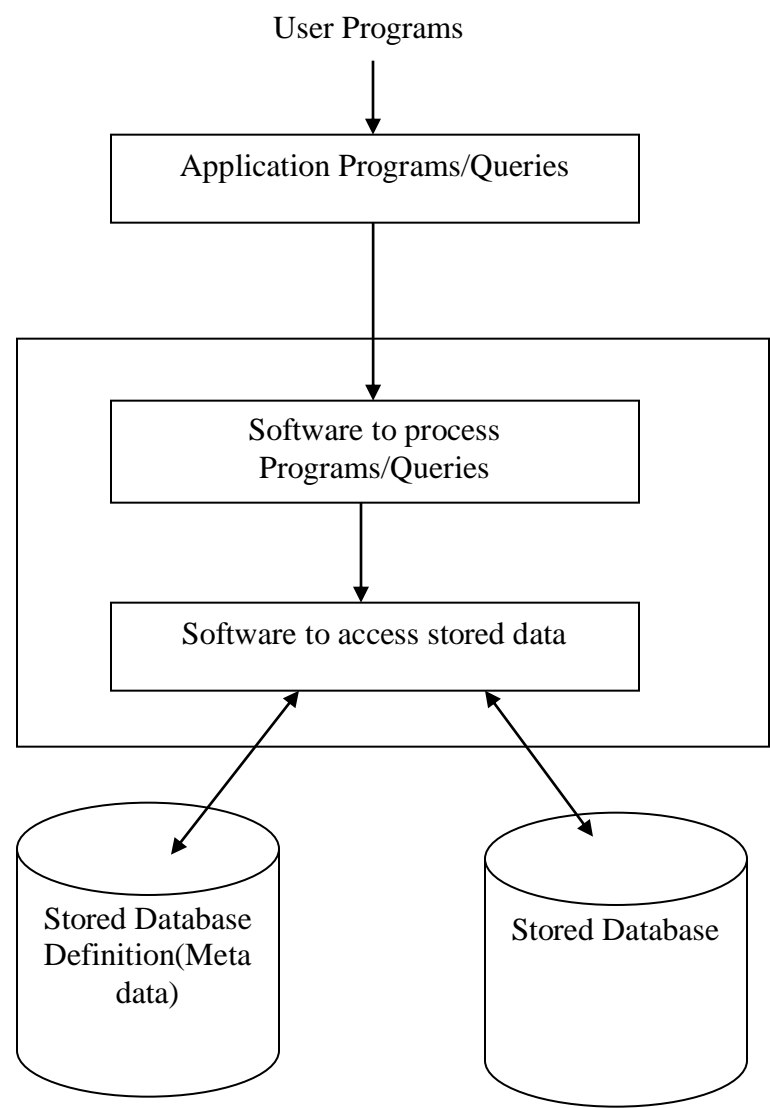
The functions performed by DBMS are as follows

- Defining a database involves specifying the data types, structures and constraints for the data to be stored in the database
- Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS
- Manipulating the database includes functions like querying the database to retrieve specific data, updating the database so as to reflect changes and generating reports from the data
- Sharing the database allows multiple users and programs to access the database concurrently
- DBMS protects the database. Protection includes both system protection against hardware or software malfunction and security malfunction against unauthorized access
- A large database may have a life cycle for many years. Hence the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

Hence DBMS is a very complex software system.

1.2 Example:

Consider a COLLEGE database maintaining information concerning students, courses and their grades in a particular college. The database is organized into three files namely STUDENT, COURSE and GRADE REPORT. To define the database, we must specify the structure of the records of each file by specifying the different data types of data elements to be stored in each record. Each student record includes the following – Roll no, name, class and department. Similarly, records in course file includes – course name, course number and department. Grade file includes roll no, class and grade. The information stored in each file is called as record. The record in Student could be {1, AMAR, IBCA, Computers}. A record in course file could be {BCA, BC001, Computers}. A record in grade file would be {1, IBCA, ‘A’}



1.3 Characteristics of Database Approach:

In file processing, each user defines and implements the files needed for a specific software application as a part of programming the application.

E.g. Consider a student file. A person trying to grade the students uses the student file and calculates grade. A clerk who wants to print students' attendance will create another file with the same data.

Hence, when files are used there is redundancy in storing the same data and redundant effort to maintain the common data. In database approach, a single repository of data is maintained that is defined and then accessed by various users.

The characteristics of database approach vs. file processing are as follows

1. Self describing nature of database system
2. Insulation between data and programs and data abstraction
3. Support of multiple views of data
4. Sharing of data and multi-user transaction processing

1.3.1 Self describing nature of database system:

A fundamental characteristic of database approach is that the database system contains not only the database but also a complete definition or description of the database structure and constraints. The definition is stored in a DBMS catalog. The catalog contains information such as structure of each file, the type and storage format for each data item and various constraints on data. The information stored in the catalog is called the **meta-data** and it describes the structure of primary database. This is used only by the DBMS users and DBMS software who need information about the database structure.

1.3.2 Insulation between data and programs and data abstraction:

The structure of data files is stored in DBMS catalog separately from access programs. This property is called **program-data independence**. The structure of the data file can be modified to reflect any changes in the data description.

Program-operation independence is a term where users define operations on data as a part of database operation. An operation is specified in 2 parts- the interface of an operation includes the name of the operation and data types of the parameters. The implementation of the operations is specified separately without affecting the interface. The characteristic that allows both program-data independence and program-operation independence is called **data abstraction**.

DBMS provides user with conceptual representation of data that does not include how data is stored or how methods are implemented. In database approach, the detailed structure and organization of the file are stored in a catalog. Database users and applications only refer to the conceptual representation.

1.3.3 Support of multiple views of data:

A database typically has many users, each of whom may require a different view or perspective of the database. A view may be a subset of the database or may contain virtual data that is derived from database files. A multi-user DBMS whose users have a variety of distinct applications must provide facility for defining multiple views.

1.3.4 Sharing of data and multi-user transaction processing

A multi user DBMS must allow multiple users to access the database at the same time. DBMS must include concurrency control software to ensure that several users trying to update the same data in a controlled manner. DBMS must support multiple users to make concurrent transactions. A transaction is an executing program or a process that includes one or more database accesses, such as reading or updating a database records. Each transaction is supposed to execute a logically correct database. The two most important properties of transactions are

1. **Isolation:** A transaction appears to execute in isolation from other transaction even though hundreds of transactions execute concurrently.
2. **Atomicity:** This ensures that either all the database operations in a transaction are executed or none are

1.4 DBMS Users:

The various classes of database users are

- DBA
- DB Designers
- End Users
- Software engineers

DBA: DBA stands for Data Base Administrator. In a database environment, the primary resource is the database itself and the secondary resource is the database itself. Administering these resources is the responsibility of the DBA. The DBA is responsible for authorizing access to the database, coordinating and monitoring its use and acquiring the software and hardware resources as needed. DBA is also accountable for system security.

DB Designers: These are people who are responsible for identifying data to be stored in the database and for choosing appropriate structures to represent and store data

End Users: End users are people whose jobs require access to the database for querying, updating and generating reports. These people make use of the existing database.

Software Engineers: This class of users can be classified as follows

- System analyst: People who determine the requirements of the end users
- Application programmers: These people implement the above specification as programs, then test and debug and maintain the software for which the database was designed

1.5 Advantages of DBMS:

The advantages of DBMS approach are as follows

1. **Controlled Redundancy:** In traditional file processing each user group maintain their own files and handle data processing application. This leads to a stage where the same data is stored in multiple files. The redundancy of storing data in more than one file creates a lot of problems
 - Entering a new record should be done on multiple files leads to duplication of effort
 - Storage space is wasted since the same details appear in all the files

- **Data inconsistency-** if a record is modified in one file this updation does not reflect in other files

In order to overcome these limitations, the database approach integrates all the views of the different user group during database design. All the required details are stored in the database in one place. This ensures consistency and saves storage space. But, in practice, only some data items may repeat themselves at the required places. This phenomenon is called controlled redundancy.

- 2. Restricting unauthorized access:** When multiple users share a large database, it is likely that all users will not want all the information in the database. Moreover, all the users will not be authorized to use all the data. Some users may be only allowed to retrieve while others may be allowed to retrieve as well as to update data. In order to provide authorization, users are given user names and passwords. A DBMS provides security and authorization subsystem, which the DBA uses to create accounts and passwords to different users
- 3. Providing Storage structures for efficient query processing:** Database systems must provide capabilities for efficiently executing a query. DBMS must provide specialized data structures to speed up disk access. The query processing and optimization module of the DBMS is responsible for choosing efficient query execution plan for each query based on the existing storage structures
- 4. Providing Backup and recovery:** DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery systems are responsible for recovery of data in such situations
- 5. Providing multiple user interfaces:** DBMS has a wide variety of users. Hence, it should provide multiple user interfaces for each class of user. For example: Query language for casual users, programming language interface for application programmers.
- 6. Represent complex relationships among data:** A database may contain a variety of data that are interrelated. DBMS must provide a capability to represent complex relationship among data as well as to retrieve and update related data only.
- 7. Enforcing integrity constraints:** DBMS should provide capability for defining and enforcing integrity constraints. For example specifying data types for each data item is done to ensure that erroneous data is not entered into the data base
- 8. Permitting inferencing and actions using rules:** Some database provide capabilities for defining deduction rules for inferencing new information from the stored database facts. Such systems are called deductive database systems. Some database provide active rules that can automatically initiate actions when certain events and conditions occur.

1.5.1 Additional Implications of using database approach:

Some of the added benefits of database approach are

- Potential for enforcing standards- Database approach permits DBA to define and enforce standards among database users in large organizations. These rules include format of data elements, display formats and report structures.
- Reduced Application Development time- Developing an application using DBMS requires very little time.
- Availability of up-to-date information- DBMS makes the database available to all the users. Once any change is made to the database, these changes can be viewed by all the users.
- Flexibility- DBMS facilitates changes in structure of database as the requirements change.
- Economies of scale: DBMS reduces the overall cost of operation and management.

Assignment

Short Answers (2 marks)

1. Define database
2. Define DBMS
3. What is the role of DBA?
4. List the different users of databases
5. Who are end users in database approach?

Long answers (4 or more marks)

1. Explain the advantages of database approach
2. Explain the various database users
3. Explain the characteristics of database approach.
4. Explain the functions of DBMS.

CHAPTER 2

DATABASE SYSTEMS - CONCEPTS AND ARCHITECTURE

2.1 Data Models:

One fundamental characteristic of database approach is that it provides some level of data abstraction by hiding details of data storage that are not needed by most users. A **data model** is a collection of concepts that can be used to describe the structure of database and provides necessary means to achieve this abstraction. The structure of database means the data types, relationships, and constraints that should hold good for the data. Data models include a set of basic operations that are used for specifying updates and retrieval on the data base. In addition to the basic operations data models are also used to specify the dynamic aspect or behavior of the database application. This allows database designers to specify a set of valid user defined operations that are allowed on the database.

2.1.1 Categories of data models:

Firstly, data models can be categorized according to the types of concepts they use to describe the database structure.

1. High-Level or conceptual data models- provide concepts that are close to the way how many users perceive data
2. Low-Level or physical data models – provide concepts that describe details of how the data is stored in computers.

Between these two extremes is a class of representational data models, which provide concepts that may be understood by the end users and also specifies details about data storage. Representational data models can be categorized as

- a. Hierarchical model
- b. Network model
- c. Relational model

Representational data models represent data using their record structures.

Hierarchical Data Model: Here different records are interrelated through a hierarchical or a tree-like structure. A parent record can have several children but a child can have only one parent.

Network Data Model: Here, a parent record can have several children and a child record can also have many parent records.

Relational Data Model: In this model, there are no physical links between records. All the data is maintained in the form of tables, comprising of rows and columns. Data in two tables is related through common columns. Querying is much easier in this model. It is very much programmer friendly and is the most widely used model.

2.2 Schemas and Instances

Schema: The description of database is called a database schema which is specified during database design and is not expected to change. A diagram used to display schema is called a **schema diagram**. Each object in the schema is called a **schema construct**. The data in the

database at a particular instant of time is called a **database state** or a **snapshot**. It is also called the current set of occurrences or **instances** in the database.

STUDENT

Roll No	Student Name	Class	Department
---------	--------------	-------	------------

COURSE

Course Name	Course Number	Department
-------------	---------------	------------

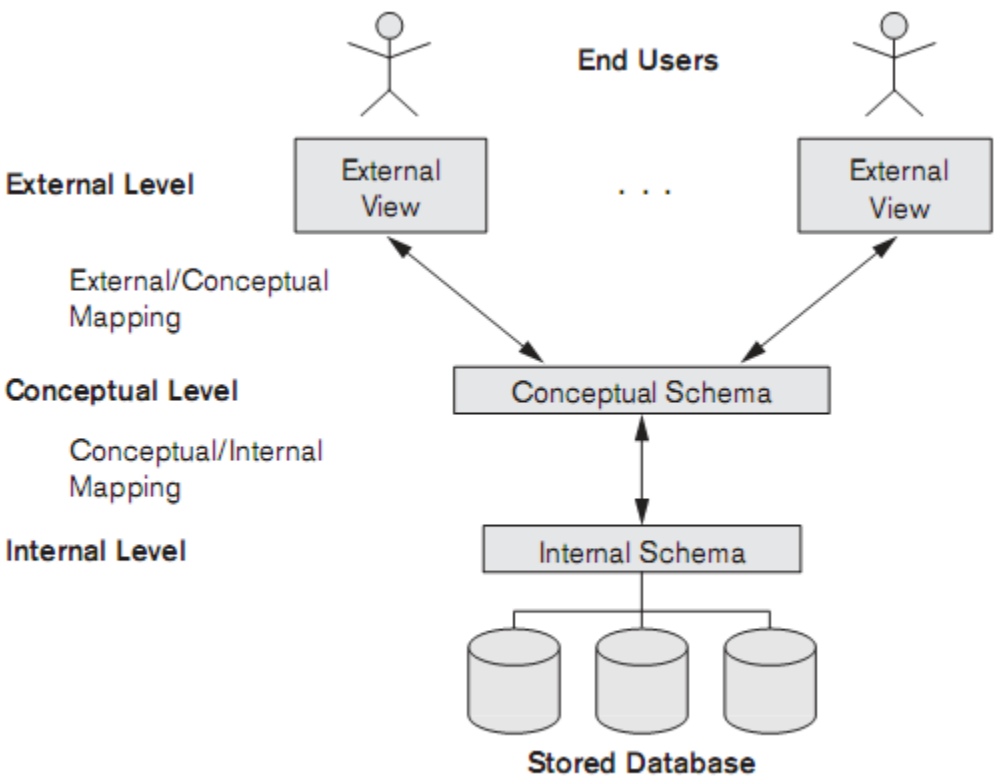
GRADE REPORT

Roll No	Class	Grade
---------	-------	-------

Meta data: The description of the schema constructs and constraints is called meta data. The meta data is stored in the DBMS catalog so that DBMS software can refer to it whenever necessary.

2.3 Three Schema Architecture:

The goal of three schema architecture is to separate user applications and the physical database. The schemas are defined at the following three levels



The internal level has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses physical data model and describes the complete details of data storage and access paths for the database.

The conceptual level has a **conceptual schema** which describes the structure of the whole database for a community of users. It hides details of physical storage and concentrates on describing entities, data types, relationships, operations and constraints. The representational model is used to describe the conceptual schema.

The external or view level includes a number of external schemas or user views. Each external schema describes a part of the database that a particular user group is interested and hides the rest of the database from that user group.

The process of transforming requests and results between levels is called **mappings**.

2.4 Data Independence:

Data independence can be defined as the capacity to change the schema at one level of the database system without having to change the schema at the next higher level. There are two types of data independence logical data independence and physical data independence.

Logical Data Independence: It is the capacity to change the conceptual schema without having to change external schemas or application programs. We may need to change the constraints or reduce the database. After the conceptual schema undergoes logical reorganization, application programs that reference the external schema constructs should work as before.

Physical Data Independence: It is the capacity to change the internal schema without having to change conceptual schemas. Changes to the internal schema may be needed because physical files had to be reorganized.

The three schema architecture makes it easier to achieve true data independence i.e. both physical and logical data independence.

2.5 Database Languages:

DBMS must provide appropriate languages and interfaces for each category of users.

DBMS Languages: Once the database design is over and DBMS is chosen to implement the database. The main task is to specify conceptual and internal schemas for the database and any mappings between the two. In most DBMS when no strict separation of levels is maintained, one language called the DDL (Data Definition Language) is used by the DBA and designers to define both the schema. The DBMS will have a DDL compiler whose function is to process the DDL statements in order to identify the descriptions of the schema constructs and to store the schema description in the DBMS catalog. In a DBMS where a clear separation is maintained between conceptual and internal schema, the DDL is used to specify the conceptual schema only. **Storage Definition Language** is used to describe the internal schema. In a three tier architecture, a language known as **View Definition Language** is used to specify user views and their mappings to the conceptual schema.

Manipulations in the database include retrieval, insertion, deletions and modification on data. DBMS provides a language called DML (**Data Manipulation Language**) to perform manipulations on the data in the database. There are two main types of DML. They are

- **High level or non-procedural DML:** It can be used on its own to specify complex database operations in a concise manner. Many DBMSs allow high level DML statements to be entered interactively from a terminal. They specify only what to do

rather than how to do i.e. the procedural details are not mentioned here. E.g. SQL. High level DML such as SQL can specify and retrieve many records in a single DML statement. Hence they are called set-at-time or set oriented DML. They are called declarative since high level DML specify which data to retrieve rather than how to retrieve it. High level DML used in stand-alone interactive manner is called a query language

- Low level or procedural DML: They are embedded in a general purpose programming language. This type of DML retrieves individual records or objects from the database and processes them separately. They are also called record-at-time DML. When DML commands are embedded in a general purpose programming language the language is called host language and DML is called data sub language

2.6 DBMS Interfaces:

Interfaces in DBMS are user-friendly and consists of the following.

- a. Menu based interfaces for web clients or browsers: These help the users with a list of options called menus that lead the user through formulation of requests. The user need not remember any command or syntax of the query language. The query is composed step by step by picking options from the menu.
- b. Form based Interfaces: It displays a form to each user. Users can fill data in the form to insert new data. The user can also fill specific entries in which case the DBMS retrieves data from the database
- c. Graphical user interfaces: A GUI displays a schema in a diagrammatic form to the user. The user specifies a query by manipulating the diagram. A pointing device such as mouse is used.
- d. Natural Language Interfaces: These interfaces accept request written in English and attempts to understand them. It has its own schema and dictionary of important words. If the interpretation is successful, the interface generates a high level query corresponding to the natural language request and submits it to the DBMS. Otherwise a dialog is started with the user for clarification of request.
- e. Interface for parametric users: Parametric users such as bank tellers perform small set of operations on the database repeatedly. A small set of abbreviated commands are included to minimize the number of keystrokes for each request.
- f. Interface for DBA: Privileged commands can be used only by the DBA. These include commands for creating user accounts, granting account authorization and reorganizing storage structure of data.

2.7 Database System Environment:

DBMS Component Modules:

The database and DBMS catalog is stored on disk. Access to disk is controlled primarily by the operating system which schedules disk I/O. The **stored data manager** module controls access to the DBMS information stored on the disk whether it is a part of catalog or database. Some DBMS use a **buffer manager module** that transfers the data from the disk to the main memory buffer so that it can be processed by other DBMS modules as well as application programs. The **DDL compiler** process the schema definition specified in DDL and stores the description of the schema in the catalog. The **run-time database processor** handles database

access at run time. It receives retrieval and updates operation and carries them on the database. A **query compiler** handles high level queries that are entered interactively. The **pre-compiler** extracts the DML commands from an application program and sends these commands to DML compiler for compilation into object code. The DBMS runs on a computer which can be accessed by the end users. This is known as **client program/ computer**. The database resides on a computer called the **database server**.

2.8 Classification of DBMS:

There are several criteria on which DBMS can be classified. They are

1. Based on data models
2. Based on number of users
3. Based on number of sites
4. Based on types of application.

2.8.1 Data Models: Based on data models, DBMS can be classified as

- a. Relational Data Model- data is organized as tables
- b. Hierarchical Data model- data is stored and related through tree like structures.
- c. Network Data Model- data records have a 1:N relation between them
- d. Object – relational Data Model- data is treated as objects

2.8.2 Number of users: Depending on the number of users supported by the system, DBMS can be categorized into

- a. Single user systems- that support only one user at a time
- b. Multi-user system – that support multiple users concurrently

2.8.3 Number of sites: Based on the number of sites over which the database is distributed , DBMS can be classified as

- a. Centralized DBMS- Here the data is stored at a single computer site. It supports multiple users but the DBMS and database reside at a single computer site.
- b. Distributed DBMS- The database and DBMS are distributed over many sites connected through network. This can again be divided as
 - i. Homogeneous DDBMS: They use same DBMS software at multiple sites
 - ii. Heterogeneous DDBMS: They use different DBMS software at different sites

2.8.4 Types of application: Based on the types of application DBMS can be classified as follows

- a. General Purpose DBMS: DBMS that can be used for all types of application
- b. Special Purpose DBMS: DBMS used for a specific application for which they are designed. E.g. airline reservation – it cannot be used for other applications.

Assignment

Short Answers (2 marks)

1. Define schema
2. What is a schema diagram? Give example
3. Define snapshot.

4. Define mapping
5. What is logical data independence?
6. What is physical data independence?

Long answers (4 or more marks)

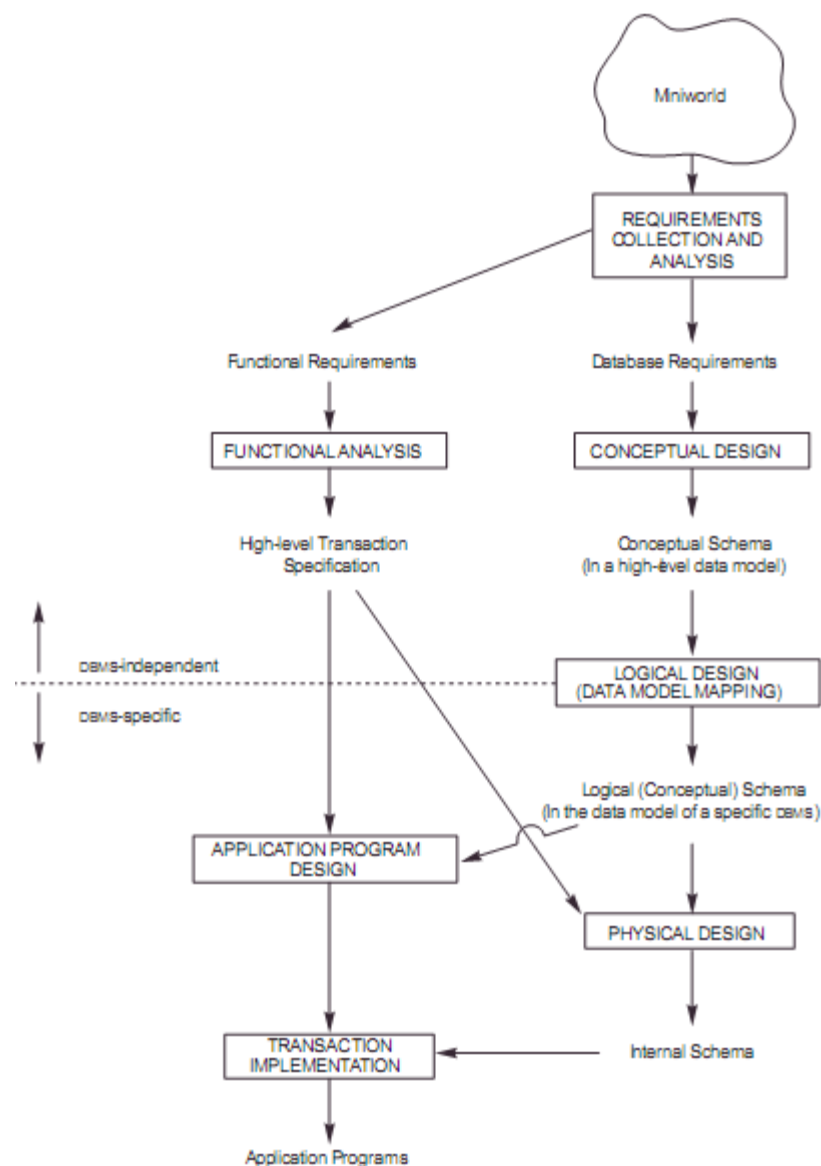
1. Explain the various categories of data models
2. Classify DBMS
3. Explain the three schema architecture with a neat diagram
4. Explain the various types of DBMS interfaces
5. Explain the various DBMS languages
6. Explain the Database system environment

CHAPTER 3

DATA MODELING USING ER MODEL

3.1 High level Conceptual Model for Database Design:

The figure below shows a simplified description of database design process.



The following are the steps used in database design process

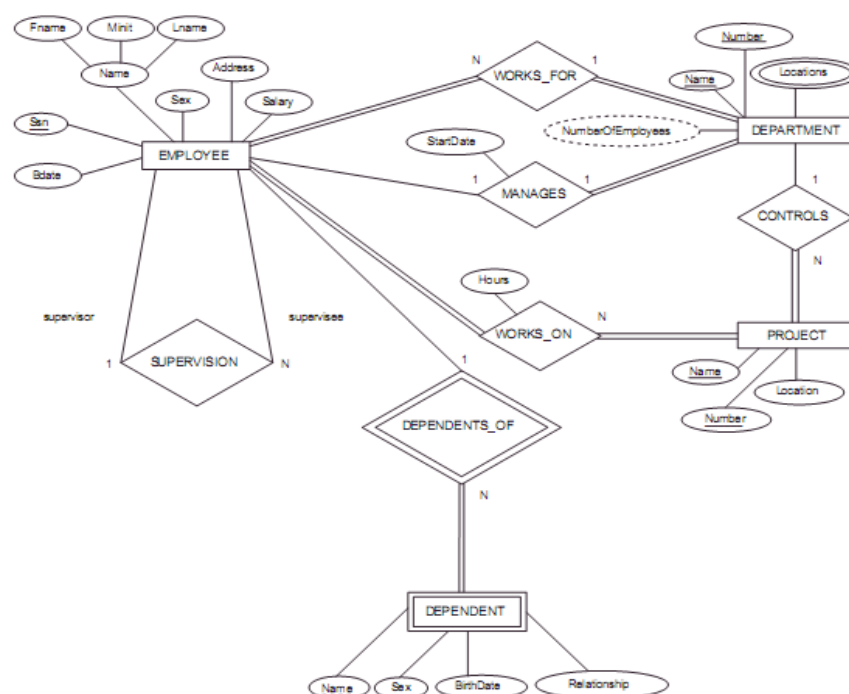
- It begins with **requirements collection and analysis**. Here the database designers interview the database users to understand and document the requirements. A written document consists of all the user requirements.

- In addition to these, the **functional requirements** of the application are also specified. They consist of the user-defined operations that will be applied to the database.
- Once the requirements are gathered, the next step is to create a conceptual schema for the database. This is called **conceptual design**.
- The conceptual design is a concise description of the requirements of the users and detailed description of the entity types, relationships and constraints.
- After the conceptual schema design, the basic data model operations can be used to specify the high level user operations identified by function analysis
- The next step is the actual implementation of the database using some data model. This step is the **logical design** or data model mapping and results in a database schema
- The last step is the physical design during which the internal storage structures and organization of the database files are specified

3.2 Database Application – Example:

Let us consider COMPANY database. This database keeps track of its employees, department and projects. After requirements collection and analysis the database designers represent the company details as follows:

- The company is organized into departments. Each department has a unique number, name and an employee who manages the department
- A department controls many projects, each of which has a unique name, number and location
- We store each employee's name, SSN, address, salary sex and date of birth.
- We keep track of each employee's dependent for insurance purpose. We keep track of dependent's name, sex, birth date and relationship with the employee.



3.3. Entity Types, Entity Sets, Attributes and Keys

ER model describes data as entities, relations and attributes.

3.3.1 Entity: The basic object of ER model is an entity. An entity can be described as a ‘thing’ in real world. An entity may be an object with physical existence, like person, employee, student or it could be an object with conceptual existence like company, job, and course. Each entity has **attributes** – particular properties that describe it. A particular entity will have a value for each of its attributes. The attribute values that describe the entity become major part of the data stored in database.

E.g. Employee is an entity. It can have the following attributes

Employee name, DOB, DOJ, designation

“John”, 19-05-1981, 21-6-2007, “Officer” is a set of values assigned to the attributes.

3.3.2 Types of Attributes:

The different types of attributes that occur in ER model are:

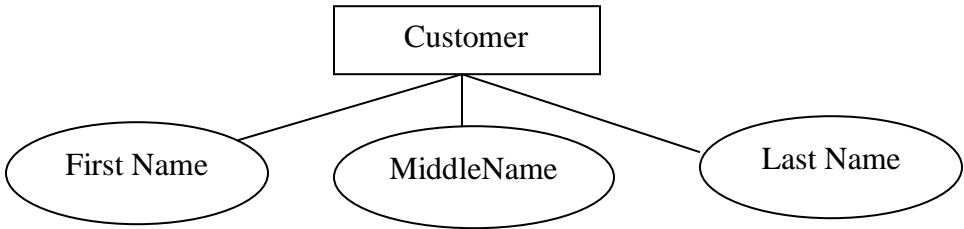
- a. Simple Vs. composite attributes
- b. Single valued Vs. Multi valued attributes
- c. Stored Vs. Derived Attributes
 - a. Simple Vs. Composite attributes: A **composite attribute** can be divided into smaller subparts which represent more basic attributes with independent meanings. Attributes that are not divisible are called **simple attributes or atomic attributes**.
 - b. Single valued Vs. Multi valued attributes: Attributes that can have a single value for a particular entity are called **single valued attributes**. E.g. Age of an employee can have only single value. An attribute that can have a set of values for the same entity is called **multi valued attribute**. E.g. Qualification of an employee can have multiple values.
 - c. Stored Vs. Derived Attributes: When two or more attributes are related, it is possible to derive one attribute from the other. E.g. If DOB is one of the attribute, then age can be derived from DOB. Here age is the derived attribute and DOB is the stored attribute.
 - d. Null Values: An entity may not have an applicable value for an attribute or some times the value may be unknown. In such cases, a special value called null is assigned to the attributes.

3.3.3 Entity Type:

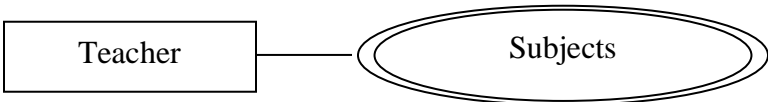
It defines a collection or set of entities that have the same attributes. Each entity type in the database is described by its name and attributes. In an ER diagram, an **entity type** is represented by a **rectangular box** enclosing the name of the entity type. The **attribute** names are enclosed in **ovals** and are attached to their entity types by straight lines. **Multi valued attributes** are described using **double ovals**. An entity type describes the schema or intension for a set of entities that share the same structure.

Examples

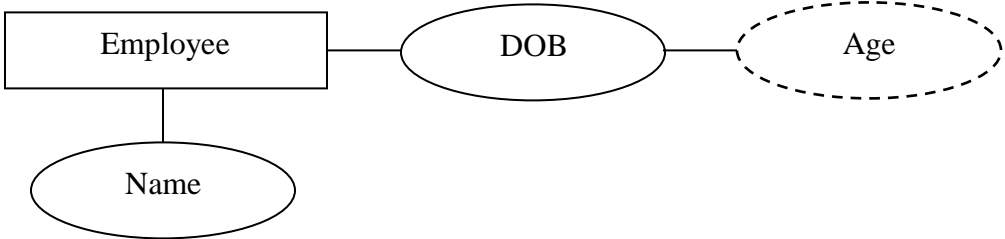
1. Composite Attributes: The attribute “customer address” can have the attributes number, street, city, and state. These are called composite attributes.



2. Multivalued Attributes: A teacher entity can have multiple subject values

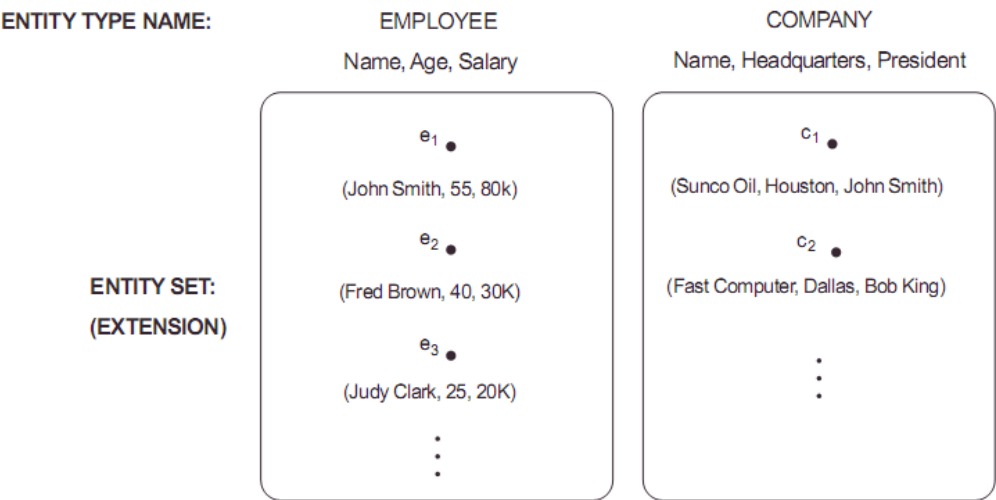


3. Derived Attributes:



3.3.4 Entity Sets:

The collection of all entities of a particular entity type in a database at a point of time is called entity set. This is also called the extension of the entity type.



3.3.5 Key Attributes of entity types:

An entity type has an attribute whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute and its value can be used to identify each entity uniquely. An entity with no key attribute is called a **weak entity**.

Domains of attributes: Each simple attribute is associated with a value set or domain of values. It specifies the set of values that may be assigned to that attribute for each individual entity. E.g. The value set for attribute age of employee to be the set of integers between 16 and 70.

3.4 Relationships types and relationship roles and Structural constraints

A relationship type R among n entity types E₁, E₂, ..., E_n defines a set of associations or relationship set among entities from these entity types. For example, a relation Employee is associated with Department. The relationship used here is “works for”. In ER diagrams, relationship types are displayed in diamond boxes, which are connected using straight lines to rectangles representing entity types. The name of the relationship type is written inside the diamond.

3.4.1 Degree of a relationship:

The degree of relationship is the number of participating entity types. The works-for relationship is of degree 2 because the two entity types participating in the relation are emp and dept. A relationship of degree two is called a binary relation and relation of degree three is called ternary relation. Binary relations are the most common relation.

3.4.2 Relationships as Attributes:

Consider the works for relation. Department of employee or employees of department represent works for relation. Hence relation can be represented as attributes.

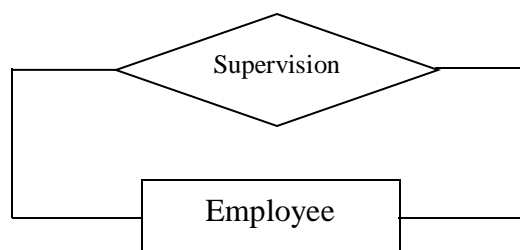
3.4.3 Role Names:

Each entity type that participates in a relation type plays a particular role in the relation. The role name signifies the role that the participating entity from the entity type plays in each relation instance and helps to explain what the relation means. Example: In ‘works for’ relation EMP plays employee or worker role and DEPT plays the employer role.

3.4.4 Recursive Relationships:

Sometimes the same entity type participates more than once in a relation type in different roles. Such a relation type is called a recursive relation.

Example: The supervise relation relates an employee to a supervisor ; where emp and supervisor are members of same EMP entity type. Here EMP entity type participates twice in ‘supervision’ – once in the role of a boss and once in the role of a subordinate.



3.4.5 Constraints on relationship types:

The two types of relation constraints are

- Cardinality Ratio
- Participation Constraints

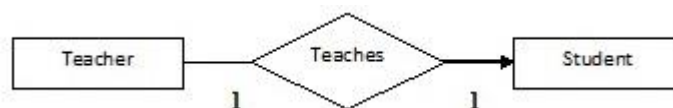
Cardinality Ratio for a Binary Relation:

The cardinality ratio for a binary relation specifies the maximum number of instances that an entity participates in. The possible cardinality ratios for a binary relationship type are 1:1, 1:N, N:1, M:N.

Example:

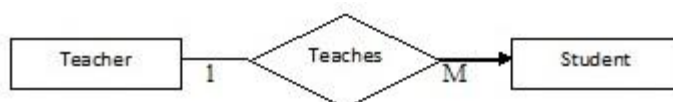
- **One-to-One**

Only one entity of the first set is related to only one entity of the second set. E.g. *A teacher teaches a student*. Only one teacher is teaching only one student. This can be expressed in the following diagram as:



- **One-to-Many**

Only one entity of the first set is related to multiple entities of the second set. E.g. *A teacher teaches students*. Only one teacher is teaching many students. This can be expressed in the following diagram as:



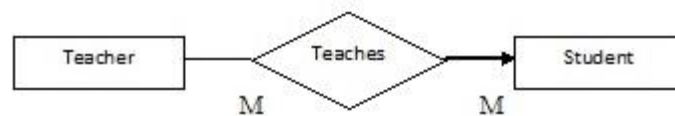
- **Many-to-One**

Multiple entities of the first set are related to multiple entities of the second set. E.g. *Teachers teach a student*. Many teachers are teaching only one student. This can be expressed in the following diagram as:



- **Many-to-Many**

Multiple entities of the first set is related to multiple entities of the second set. E.g. *Teachers teach students*. In any school or college many teachers are teaching many students. This can be considered as a two way one-to-many relationship. This can be expressed in the following diagram as:



Participation Constraints:

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via a relation type. It specifies the minimum number of relation instance that each entity can participate in and is also called the minimum cardinality relation. The two types of participation constraints are total and partial participation.

Example: If a company policy states that every employee must work for a department. An employee entity can exist only if it participates in at least one 'works for' relation instance. The participation of EMPLOYEE in 'works for' is called **total participation**. i.e. every entity in a set of employee must be related to department entity via works for. Total participation is also called **existency dependency**. Not every employee can manage a department. Hence participation of employee in 'manages' relation is partial i.e. a part of the set of employee entities are related to some department entity via 'manages' relation. Cardinality ratio and participation constraint together are known as **structural constraints**.

3.5 Weak Entity Types:

Entity types that do not have any key attributes of their own are known as **weak entity types**. Entity types that have key attributes are called **strong entity types**. Entities belonging to a weak entity are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type as **identifying or owner entity type**. The relation type that relates the weak entity type to its owner is called the **identifying relation** of the weak entity.










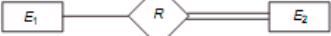
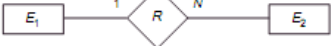
Example: Consider the entity type Dependant related to Employee, used to keep track of the dependents of each employee.

Dependent – Name, DOB, Sex, Relation.

Two dependents of 2 distinct employees may by chance be the same value. They can be identified as distinct only after determining the particular employee entity to which each dependent is related. In this example, EMP is parent entity type or dominant entity type. Weak entity is the **child/subordinate entity type**. A weak entity has a partial key which is a set of attributes that can uniquely identify weak entities that are related to the same owner entity.

In an ER diagram, weak entity and its attributes are surrounded by double lines. Partial key attributes is underlined with dashed lines.

Symbols used in ER diagram

Symbol	Meaning
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E_2 IN R
	CARDINALITY RATIO 1:N FOR $E_1:E_2$ IN R

Assignment

Short Answers (2 marks)

- 1. Define entity type
- 2. Define entity set
- 3. Define ER model
- 4. What are weak entity types
- 5. Give an example for recursive relation
- 6. What do you mean by cardinality ratio?
- 7. What do you mean by degree of a relation? Give example
- 8. Define a recursive relation with an example.

Long answers (4 or more marks)

- 1. Explain the various types of attributes in a relation with examples
- 2. List the various symbols used in ER diagrams
- 3. Explain about the various cardinality ratios on relations.

UNIT II

CHAPTER 4

RELATIONAL DATA MODEL AND RELATIONAL DATABASE CONSTRAINTS

4.1 Relational Model Concepts:

The relational model represents the database as a collection of relations. A relation is a table of values where each row represents a collection of data values.

4.1.1 Domain: A domain D is a set of atomic values. Each value in the domain is indivisible. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.

Example:

Set of telephone numbers- Numbers

Set of student names- Character strings representing names of students

They are logical definitions of domains. A data type or format is specified for each domain.

4.1.2 Relation Schema: A relation schema $R(A_1, A_2, \dots, A_n)$ is made up of a relation R and a list of attributes A_1, A_2, \dots, A_n . Each attribute A_i is the name of the role played by some domain D in the relation schema R . D is called the domain of A_i and is denoted by $\text{dom}(A_i)$. A relation schema is used to describe a relation. The degree of a relation is the number of attributes in the relation schema. A relation or a relation state r of a relation schema $R(A_1, A_2, \dots, A_n)$, is also denoted by $r(R)$ is a set of n -tuples $r = \{t_1, t_2, \dots, t_n\}$. Each n -tuple is an ordered list of values $t = \langle v_1, v_2, \dots, v_n \rangle$ where each value v_i is $1 \leq i \leq n$ is an element of $\text{dom}(A_i)$ or a special null value.

Tuple: Each row in a relation is called a tuple.

Attribute: The column header in a relation is called an attribute of a relation.

4.1.3 Characteristics of Relations:

1. Ordering of tuples in a relation: A relation is defined as a set of tuples. Mathematically, elements of a set do not have any order. Hence the tuples in a relation do not have any order. When they are stored physically on a disk, there is always an order. Hence when a table is displayed, it is displayed in the order it is stored. But, the order can be explicitly mentioned, when the relation is displayed.
2. Ordering of values within a tuple: A n -tuple relation is an ordered list of n -values. Hence ordering of values in a tuple is important.

Relation: A relation schema $R = \{A_1, A_2, \dots, A_n\}$ is a set of attributes and the relation state $r(R)$ is a finite set of mappings $r = \{t_1, t_2, \dots, t_m\}$ where each tuple t_i is a mapping from R to D . Here D is the union of attributes domains. i.e. $D = \text{dom}(A_1) \cup \text{dom}(A_2) \cup \dots \cup \text{dom}(A_n)$. Each mapping t_i is called a tuple. A tuple can be considered as a set of ($\langle \text{attribute} \rangle, \langle \text{value} \rangle$) pairs where each pair gives a value of a mapping from attribute A_i to a value v_i from $\text{dom}(A_i)$

3. Values and nulls in a tuple: Each value in a tuple is an atomic value. This model is called a flat relation. Hence composite and multi-valued attribute are not allowed. Multi-valued attributes must be represented by a separate relation. Null values are used to represent the values of attributes that may be unknown or may not be applied to a tuple.

4.2 Relational Model Constraints and Database Schemas :

Constraints are restrictions imposed on the actual values in the database state. Constraints on the database can be divided into three categories.

- Model based constraints: These are constraints that are inherent in the data model.
- Schema based constraints: These are constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL
- Application based constraints: These are constraints that are expressed and enforced in application programs.

4.2.1 Schema Based constraints: These constraints include

- a. Domain constraint
- b. Key constraints
- c. Constraints as null
- d. Entity Integrity constraint
- e. Referential Integrity constraint

Domain Constraint: This constraint specifies that within each tuple the value of each attribute A must be an atomic value from the domain $\text{dom}(A)$. The data types for domain include standard numeric data types for integer and real values, fixed length and variable length strings.

Key Constraints: A relation is a set of tuples. By definition all the elements in a set are distinct. No two tuples can have the same combination of values for all attributes. Let SK be a sub set of attributes of R with the property that no two tuples have the same combination of values. Let t_1, t_2 be two distinct tuples. Then, $t_1[\text{SK}] \neq t_2[\text{SK}]$.

Such set of attributes is called a super key of the relation schema R. A super key SK specifies uniqueness constraint that no two tuples in the state r of R can have the same value for SK. Every relation has at least one default super key- the set of all its attributes. A super key can have redundant attributes. A key K of a relation R is a super key of R with an additional property that removing any attribute A from K leaves a set of attributes K' that is not a super key of R. A key satisfies two constraints:

1. Two distinct tuples in any state of a relation cannot have identical values for the attributes in the key.
2. It is a minimal super key i.e. a super key from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 to hold.

Candidate Key: A relation schema can have more than one key. Each of the key is called a candidate key.

Primary key: A candidate key whose values are used to identify tuples in a relation is called a primary key.

Constraints as null: This is a constraint on attributes that specifies whether null values are permitted or not. E.g. If every student tuple must have a valid non-null value for name attribute, then the name attribute is constrained as not null.

Entity Integrity Constraint: It states that no primary key value can be null. This is because primary keys are used to identify the tuples in a relation.

Referential Integrity Constraint: The referential integrity constraint is specified between two relations and is used to maintain consistency among tuples in two relations. This constraint specifies that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. The concept of foreign key is used to define referential integrity

Foreign Key: Let R1, R2 be two relation schemas. A set of attributes FK in a relation schema R1 is a foreign key of R1 that references a relation R2 if it satisfies the following rules

- The attributes in FK have the same domains as primary key attributes PK of R2. The attributes FK are said to refer to the relation R2.
- A value of FK in a tuple t1 of current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is null. Then we have, $t1[FK]=t2[PK]$. We say that t1 refers to the tuple t2.

Here R1 is called the referencing relation and R2 is called the referenced relation. If these conditions hold, a referential integrity constraint from R1 to R2 is said to hold.

4.3 Operations on Relations: Violation of Constraints

4.3.1 Insert operation: It provides a list of attribute values for a new tuple that is to be inserted into R. Insert operation can violate the following constraints:

- Domain constraint can be violated if an attribute value does not match the specified domain.
- Key constraint can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).
- Entity integrity constraint can be violated if the primary key of the new tuple t is null
- Referential integrity constraint can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

4.3.2 Delete Operation: This operation deletes a tuple from a relation.

- It can violate only referential integrity constraint. This occurs in case the tuple being deleted is referenced by foreign key from other tuples in the database.

4.3.3 Update Operation: This operation is used to modify / change values of one or more attributes in tuple(s) in a relation R. It is necessary to specify a condition on attributes of a relation to select a tuple to be modified. Updating an attribute that is neither a primary key nor a foreign key creates no problem. Modifying a primary key is similar to deleting a tuple and inserting another in its place. When updating DBMS checks to confirm the new value is of correct data type and domain.

Assignment

Short Answers (2 marks)

1. Define primary key
2. Define candidate key
3. Define foreign key
4. Define domain of a relation
5. Define relation schema
6. Define tuple in a relation
7. Define attribute in a relation

Long answers (4 or more marks)

1. Explain Entity and referential integrity constraints.
2. Explain the characteristics of relation
3. Explain the schema based constraints on a relation.

CHAPTER 5 RELATIONAL ALGEBRA

The basic set of operations on relational model is the relational algebra. These operations enable a user to specify basic retrieval requests. The result of retrieval is a new relation that is formed from one or more relations

5.1 Unary Relational Operations:

The operations that operate on single relation are called unary operations. The unary operations in relational algebra are

a) Select

b) Project

5.1.1 SELECT Operation: The select operation is used to select a subset of tuples from a relation that satisfy a selection condition. It can be considered as a filter that keeps only those tuples that satisfy a qualifying condition. The select operation can be visualized as a horizontal partition of the relation into two sets of tuples- those that satisfy the condition are selected and those that do not satisfy the condition are discarded. Select operation is denoted by

$$\sigma_{\langle \text{select condition} \rangle}(R)$$

E.g. To select employee tuples whose department is 4, it can be specified as follows:

$$\sigma \rightarrow \sigma_{\text{DNO}=4}(\text{EMPLOYEE})$$

The boolean expression specified in the selection condition is made up of a number of clauses of the form

<attribute name> <comparison operator> <constant value>

OR

<attribute name> <comparison operator> <attribute name>

The comparison operators can be any of the elements in the set $\{=, \neq, \leq, \geq, <, >\}$

Two or more selection conditions can be combined using boolean operators like AND, OR and NOT.

Condition 1 AND condition2 is true only if both condition1 and condition2 are true, otherwise it is false.

Condition 1 OR condition2 is true only if either of the condition is true, otherwise it is false.

NOT condition is true if condition is false and false otherwise

The select operation is commutative

$$\text{i.e. } \sigma_{\text{condition 1}} (\sigma_{\text{condition 2}} (R)) = \sigma_{\text{condition 2}} \sigma_{\text{condition 1}} (R)$$

A sequence of selects can be applied in any order. We can also combine a cascade of select operations into a single select operation with an AND condition

$$\text{i.e. } \sigma_{\text{condition 1}} (\sigma_{\text{condition 2}} (\dots \sigma_{\text{condition n}} (R))) = \sigma_{\text{condition 1 AND condition 2 AND condition 3} \dots \text{condition n}}(R)$$

5.1.2 PROJECT Operation: Project operation selects certain columns from the table and discards other columns. It can be visualized as vertical partitioning of relation into two relations – one has the required attributes that contains the result of the operation and the other contains discarded columns. Project operations is denoted by

$$\Pi_{\langle \text{attribute list} \rangle} (R)$$

Here, <attribute list> is the desired list of attributes that are to be projected from the relation R

For example, if the name and salary of an employee is to be listed, it can be written as follows:

$\Pi_{NAME,SAL}(EMPLOYEE)$

The number of tuples in a relation resulting from project operation is always less than or equal to the total number of tuples in R.

5.2 Set Theory Operations:

Standard mathematical operations on sets can also be applied to relational algebra. The three operations in this category are

- a) UNION
- b) INTERSECTION
- c)MINUS

Union Compatibility:

These are binary operations and are applied on two relations. But, the relations must have same type of tuples. Two relations $R(A_1,A_2,...,A_n)$ and $S(B_1,B_2,...,B_n)$ are said to be union compatible if they have the same degree n and if $dom(A_i)=dom(B_i)$ for $1 \leq i \leq n$

i.e. Two relations should have the same number of attributes and each corresponding pair of attributes should have the same domain. This condition is known as **union compatibility**.

Example: Two union Compatible relations

STUDENT

FNAME	LNAME
Suresh	Rao
Ramesh	Krishna
Ravi	Reddy
Vipul	Kumar
Vinay	Kumar
Sachin	Kumar

INSTRUCTOR

FNAME	LNAME
Sachin	Kumar
Rohit	Sharma
Ravi	Reddy

Let R and S be two union compatible relations. The given set of binary operations can be defined as follows:

5.2.1 UNION – The result of this operation is denoted by $R \cup S$. The result is a relation that includes all tuples that are in R or in S or in both R and S. Duplicate tuples are eliminated.

Example: $STUDENT \cup INSTRUCTOR$

FNAME	LNAME
Suresh	Rao
Ramesh	Krishna
Ravi	Reddy
Vipul	Kumar
Vinay	Kumar
Sachin	Kumar
Rohit	Sharma

5.2.2 INTERSECTION – The result of this operation is denoted by $R \cap S$. It results in a relation that contains all tuples that are present in both R and S

Example: $STUDENT \cap INSTRUCTOR$

FNAME	LNAME
Ravi	Reddy
Sachin	Kumar

5.2.3 SET DIFFERENCE – The result of this operation is denoted by $R - S$. It is a relation that includes all tuples that are in R but not in S.

Examples:

a) $STUDENT - INSTRUCTOR$

FNAME	LNAME
Suresh	Rao
Ramesh	Krishna
Vipul	Kumar
Vinay	Kumar

b) $INSTRUCTOR - STUDENT$

FNAME	LNAME
Rohit	Sharma

5.2.4 Cartesian Product:

It is also called cross product or cross join and is denoted by \times . This is a binary set operation. It is used to combine relation in a combinatorial fashion. If a relation R has n tuples and relation S has am tuples, the total number of tuples in $R \times S$ is $m \times n$. It creates tuples with combined attributes of two relations. We can select only related tuples from two relations by specifying appropriate selection conditions.

5.3 Binary Relational Operators:

5.3.1 Join Operation:

The join operation denoted by \Join is used to combine related tuples from two relations into a single tuple. This operation is used to process relationships among relations. The general form of join operation on two relations $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, B_3, \dots, B_m)$ is $R \Join_{(join\ condition)} S$. The result of the above operation is a relation Q with $n+m$ attributes $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$. Q has one tuple for each combination of tuples one from R and one from S- whenever the combination satisfies the join condition. The main difference between Cartesian product and join is that in Cartesian product all combinations of tuples are included whereas in the latter case the result contains tuples that satisfy the join condition. The different types of join are

a) Equi Join b) Natural Join c) Theta Join

a) **Equi-Join** – When join conditions involve only equality comparisons on the attributes of 2 tables ,then, such joins are called equi-joins. The result of equi-join operation has always one or more pair of attributes that are identical in every tuple.

EMP

EMPNO	ENAME	DNO
E001	RAMA	D001
E002	GITA	D002
E003	RITA	D001

DEPT

DNUM	DNAME
D001	HR
D002	SALES

EMP⋈ DEPT

EMPNO	ENAME	DEPTNO	DNAME
E001	RAMA	D001	HR
E002	GITA	D002	SALES
E003	RITA	D001	HR

b) **Natural Join**- In order to eliminate identical values in every tuple, we use natural joins. The definition of natural join requires that the two join attributes have the same name in both the relations. It is denoted by. In case the two attributes do not have the same name, renaming is done.

EMP

EMPNO	ENAME	DNAME
E001	RAMA	HR
E002	GITA	SALES
E003	RITA	HR

DEPT

DNAME	DLOCATION
HR	CHENNAI
SALES	KANNUR

EMP⋈ DEPT

EMPNO	ENAME	DNAME
E001	RAMA	HR
E002	GITA	SALES
E003	RITA	HR

c) **Theta Join** – When the join conditions involve all comparison operators on the attributes i.e. {<,>,<=,>=,!=} then, such a join is called theta join.

CAR

CarModel	CarPrice
CarA	20,000
CarB	30,000
CarC	50,000

BOAT

BoatModel	BoatPrice
Boat1	10,000
Boat2	40,000
Boat3	60,000

CAR⋈_{CARPRICE>=BOATPRICE} BOAT

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20,000	Boat1	10,000
CarB	30,000	Boat1	10,000
CarC	50,000	Boat1	10,000
CarC	50,000	Boat2	40,000

Outer Join Operations:

The join operations where only matching tuples are kept in the result are called inner joins. In inner joins, tuples are without a match and those with null values are eliminated. This amounts to loss of information if the result of the join is suppose to contain all the information. A set of operations called outer join can be used when we need all the tuples in R or all those in S or all those both in R and S, regardless of whether they have matching tuples in the other relations. This will satisfy the need of the query in which tuples from both the tables are to be combined by matching corresponding rows but without losing any tuples for lack of matching values. The different types of outer joins are

- a. Left Outer Join
- b. Right Outer Join
- c. Full Outer Join

Left Outer Join ⋈_L: This operation keeps all the tuples in the first or left relation (R x S) i.e. R. If no match is found in S then the join result pads the attribute with null values.
Employee

Name	EmpId	DeptName
Hari	3415	Finance
Samit	2241	Sales
Geetha	3401	Finance
Haritha	2202	Sales
Tom	1123	Executive

Dept

DeptName	Manager
Sales	Haritha
Production	Charles

Employee ⋈ Dept

Name	EmpId	DeptName	Manager
Hari	3415	Finance	NULL
Samit	2241	Sales	Haritha
Geetha	3401	Finance	NULL
Haritha	2202	Sales	Haritha
Tom	1123	Executive	NULL

Right Outer Join(⋈): This operation keeps all the tuples in the second or right relation (R x S) i.e. S. If no match is found in R then the join result pads the attribute with null values.

Employee

Name	EmpId	DeptName
Hari	3415	Finance
Samit	2241	Sales
Geetha	3401	Finance
Haritha	2202	Sales
Tom	1123	Executive

Dept

DeptName	Manager
Sales	Haritha
Production	Charles

Employee ⋈ Dept

Name	EmpId	DeptName	Manager
Samit	2241	Sales	Haritha
Haritha	2202	Sales	Haritha
NULL	NULL	Production	Charles

Full Outer Join(⋈): This operation keeps all the tuples in both the left and right relations. When no match is found in either relations, the corresponding attributes are padded with null values as needed.

Employee

Name	EmpId	DeptName
Hari	3415	Finance
Samit	2241	Sales
Geetha	3401	Finance
Haritha	2202	Sales
Tom	1123	Executive

Dept

DeptName	Manager
Sales	Haritha
Production	Charles

Employee ⋈ Dept

Name	EmpId	DeptName	Manager
Hari	3415	Finance	NULL
Samit	2241	Sales	Haritha
Geetha	3401	Finance	NULL
Haitha	2202	Sales	Haritha
Tom	1123	Executive	NULL
NULL	NULL	Production	Charles

5.3.2 Division Operator:

The division operation is denoted by \div and is useful for special kind of query that sometimes occur in database operation. The division operator is applied to two relations $R(Z) \div S(X)$ where $X \subseteq Z$. For a tuple t to appear in the result T of the division, the values in t must appear in R in combination with every tuple t in S .

Example: $T = R \div S$

R

Empno	Pno
101	1
102	1
103	1
104	1
101	2
103	2
102	3
103	3
104	3
101	4
102	4
103	4

S

Empno
101
102
103

T

Pno
1
4

Summary of Relational Algebra Operations:

Operations of Relational Algebra		
OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{\langle \text{selection condition} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R , and removes duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$, OR $R_1 \bowtie_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 \star_{\langle \text{join condition} \rangle} R_2$, OR $R_1 \star_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$ OR $R_1 \star R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2 ; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2 .	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

5.4 Additional Relational Operations:

5.4.1 Aggregate Functions and Grouping

Functions that are used on a collection of values from the database are called aggregate functions. Example: Retrieving average salary or total salary of employees. The most common functions applied to collection of numeric values are SUM, AVERAGE, MINIMUM, MAXIMUM, COUNT. The count function is used to count the tuples. Another request involves grouping the tuples in a relation by the value of some of their attributes and then, applying

aggregate function independently on each group. Example: To group employee tuples by deptno so that each group includes tuples for employees working in the same department. Aggregate functions are defined using the operator \mathfrak{A} and specifies the following request.

$$\langle \text{grouping attributes} \rangle \mathfrak{I} \langle \text{function list} \rangle (R)$$

where 1: list of attributes of relation specified and 2: list of (function, attribute) pair in R
 ρ is used for rename operation.

Example: To retrieve the department number, number of employees in the department and their average salary, we can write it as

$$\rho_{R(DNO, NO-OF-EMP, AVG-SAL)}(DNO \text{ } \mathfrak{Z} \text{ COUNT}_{ENO, AVERAGE_SAL}(EMPLOYEE))$$

5.4.2 Outer Union Operation:

It was developed to take union of tuples from two relations if the relations are not union compatible. This operation will take the union of two relations $R(X,Y)$ and $S(X,Z)$ that are partially compatible. The attributes that are union compatible are represented only once and those that are not are also kept in the result relation $T(X,Y,Z)$. Two tuples t_1, t_2 such that $t_1 \in R$ and $t_2 \in S$ are said to match if $t_1[X] = t_2[X]$ and are considered to represent the same entity or relation instance. These will be combined into a single tuple in T . Tuples in either relation that have no matching tuple in other relation are padded with null values.

Consider the following schemas:

STUD(Name, Eno, Dept, Adv)

INSTR(Name, Eno, Dept, Rank)

The result of STUD-OR-INSTR is

STUD-OR-INSTR(Name, Eno, Dept, Adv, Rank)

Assignment

Short Answers (2 marks)

1. What is the purpose of SELECT operation in relational algebra?
2. What is the purpose of PROJECT operation in relational algebra?
3. List the set theory operations in relational algebra.
4. What is union compatibility?
5. What do you mean by outer union operation?

Long answers (4 or more marks)

1. Explain **SELECT** operation in relational algebra.
2. Explain **PROJECT** operation in relational algebra.
3. Explain set theory operations in relational algebra.
4. Explain the various types of join operations in relational algebra.

CHAPTER 6
FUNCTIONAL DEPENDENCIES AND NORMALIZATION

6.1 Functional Dependency:

Given a relation R, a set of attributes X in R is said to functionally determine another attribute Y in R $X \rightarrow Y$ if and only if each value of X is associated with one value of Y. Here X is called determinant and Y is called dependent.

Example: Student table

Rollno	Name	Sub-code	Subject	Marks
121	Priya	BCA01	DBMS	59
121	Priya	BCA02	OOPS	70
122	Rama	BCA01	DBMS	80
122	Rama	BCA02	OOPS	65
123	Payal	BCA03	NETWORKS	70

Here, rollno does not uniquely identify rows in a table, therefore it cannot be a primary key. Similarly, sub-code does not uniquely identify rows in a table. But, a combination of rollno and sub_code uniquely identifies a row in the table. Hence (rollno,sub-code) together will be a primary key in a table

6.2 Normal Forms based on Primary Keys

Normalization: Normalization is a technique used to reduce redundancy in tables. It is a formal process for deciding which attributes should be grouped together in a relation. It serves as a tool for validating and improving logical design, so that logical design avoids unnecessary duplication of data i.e. it eliminates redundancy and promotes integrity.

Normal Forms:

The different forms of normalization that can be applied to relations are as follows:

- First Normal Form 1NF
- Second Normal Form 2NF
- Third Normal Form 3NF
- Boyce-Codd Normal Form BCNF

First Normal Form (1NF)

A relation R is said to be in 1NF if every attribute of R takes only single atomic values. In order to transform un-normalized table to 1NF we identify and remove repeating groups within a table.

Example

DEPT

Deptno	Deptname	DeptLoc
D001	Accounts	Chennai
D002	R&D	Delhi,Bangalore

The above table is transformed to 1NF as follows:

Deptno	Deptname	Deptno	DeptLoc
D001	Accounts	D001	Chennai
D002	R&D	D002	Delhi
		D002	Bangalore

6.3 Second Normal Form (2NF):

Second normal form is based on functional dependency. A relation is in 2NF if every non-prime attribute A in R is fully functionally dependent on the primary key of R.

EMP-PROJ

<u>ECODE</u>	<u>P-NUMBER</u>	ENAME	PROJ-NAME	HOURS	PLOCATION
--------------	-----------------	-------	-----------	-------	-----------

|| 2NF

E1

ECODE	P-NUMBER	HOURS
-------	----------	-------

E2

P-NUMBER	PROJ-NAME	PLOCATION
----------	-----------	-----------

E3

ECODE	ENAME
-------	-------

6.4 Third Normal Form (3NF):

Transitive Dependency:

A functional dependency $X \rightarrow Y$ is a transitive dependency if there is a set of attributes Z in R such that if $X \rightarrow Y$ and $Y \rightarrow Z$, then, $X \rightarrow Z$.

Example:

<u>ECODE</u>	ENAME	DOB	DEPTNO	DNAME	DMNGR
--------------	-------	-----	--------	-------	-------

The above table has transitive dependency. Here, $ecode \rightarrow deptno$ and $deptno \rightarrow dmngr$. Hence, $ecode \rightarrow dmngr$.

Third Normal Form (3NF):

Third Normal form is based on transitive dependency. A relation R is said to be in 3NF if it satisfies

- a) It is fully functionally dependent on every key of R
- b) It is non-transitively dependent on every key of R

Example:

EMPDEPT

<u>ECODE</u>	ENAME	DOB	ADDR	DEPTNO	DNAME	DMNGR
--------------	-------	-----	------	--------	-------	-------

3NF

EMP

<u>ECODE</u>	ENAME	DOB	ADDR	DEPTNO
--------------	-------	-----	------	--------

DEPT

DEPTNO	DNAME	DMNGR
--------	-------	-------

6.5 Boyce-Codd Normal Form (BCNF)

A relation is said to be in BCNF if and only if every determinant is a candidate key. The difference between 3NF and BCNF is that for a functional dependency $X \rightarrow Y$, the 3NF allows this dependency if Y is a primary key attribute and X is not a candidate key; whereas in BCNF X must be a candidate key. Hence, BCNF is stronger than 3NF.

Example:

PRODUCT(PROD_NO,PNAME,PRICE)

Here, $PROD_NO \rightarrow PNAME, PRICE$

In the above example, PROD_NO is the candidate key. Hence the above relation is in BCNF

Assignment

Short Answers (2 marks)

1. Define normalization
2. Define 1NF
3. Define functional dependency
4. Define transitive dependency

Long answers (4 or more marks)

1. Explain 1NF with an example
2. Explain 2NF with an example
3. Explain 3NF with an example
4. Explain BCNF with an example

CHAPTER 7

DISK STORAGE

7.1 Introduction:

The collection of data that makes up a computerized database must be physically stored on a computer storage medium. The DBMS can then retrieve, update and process data as and when needed. The two main categories of computer storage media are

- a. Primary Storage – This category includes storage media that can be directly accessed by CPU. It provides fast access to data but is of limited storage. Example: RAM, ROM
- b. Secondary Storage – Data stored on these devices cannot be directly accessed by the CPU. It must be first be copied onto the primary storage. These devices have larger capacity, cost less and provide slower access to data. Example: Magnetic disks, optical disks and tapes

7.2 Secondary Storage Devices:

7.2.1 Hardware Description of Disk Devices:

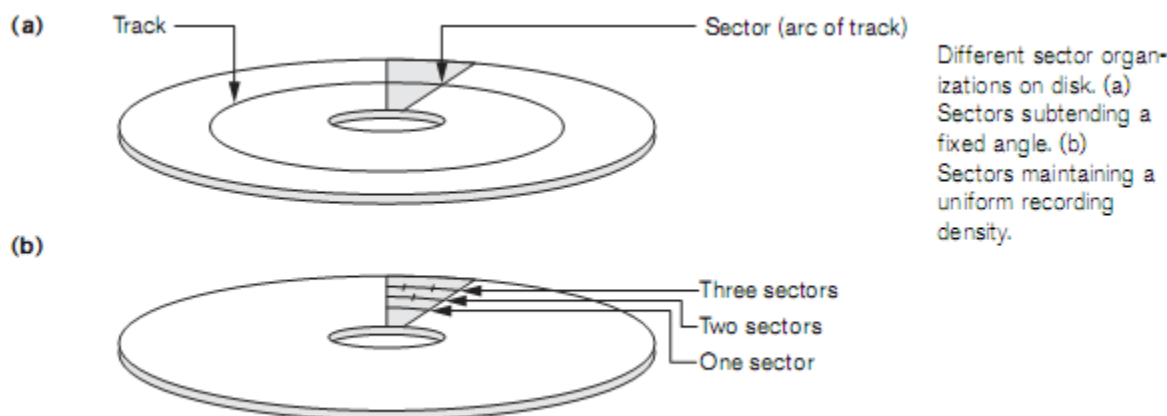
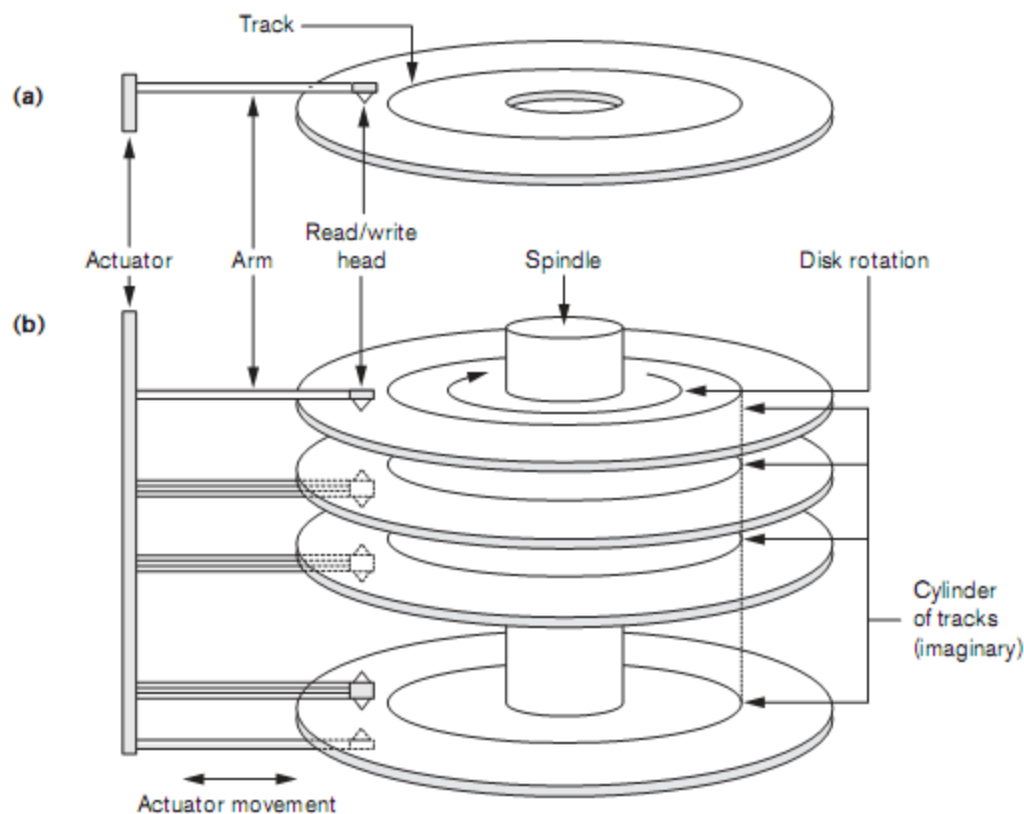
Magnetic disks are used to store large amount of data. The most basic unit of data on the disk is a single bit of information. By magnetizing an area on the disk one can represent a bit value of either a 0 or 1. To code the information bits are grouped into bytes. The capacity of the disk is the number of bytes that can be stored in a disk.

Disks are all made up of magnetic material shaped as thin circular disks and protected by a plastic or acrylic cover. A disk is single sided if it can store information only on one surface and is double sided if information can be stored on both the surfaces. To increase the storage capacity, disks are assembled into a disk a pack, which includes many disks.

Information is stored on disk surfaces in concentric circles of small width, each having a distinct diameter. Each circle is called a track. For disk packs, the tracks with same diameter on various surfaces is called **cylinder**. Each track is divided into smaller blocks called **sectors**. The division of tracks into sectors is hard coded and cannot be changed. The division of tracks into equal sized disk blocks is set by the operating system during disk formatting. A disk is a random access address device. Transfer of data between main memory and disk takes place in units of disk blocks. The physical address of a block is a combination of cylinder number, track number and block number. The address of a buffer is a contiguous reserved area in the main storage that holds one block. For a read operation, the block of data from the disk is copied into buffer. For a write operation, the block of data from the buffer is copied into disk.

The actual hardware mechanism that reads or writes a block is the read/write head, which is a part of the disk drive. Read/write heads are attached to a mechanical arm. All arms are connected to the actuator attached to an electric motor, which moves the read/write head and positions them over the required tracks. Some disk units have as many read/write heads as there are tracks on each disk. These are called fixed head disks. Disk heads with an actuator are called movable head disks. The time required for the disk controller to mechanically position the read/write head on the correct track is called **seek time**. Rotational delay/rotational latency is the time required for the beginning of the desired block to rotate and position itself on the read/write head. **Block transfer time** is the time needed to transfer data to/from the buffer and the disk.

(a) A single-sided disk with read/write hardware.
(b) A disk pack with read/write hardware.



7.2.2 Magnetic Tapes:

Disks are random access storage devices because an arbitrary disk block may be accessed at random. Magnetic tapes are sequential access devices. i.e. to access the n th block on tape, we must first scan the preceding $n-1$ blocks. A tape drive is used to read data from or to write the data to a tape reel. A read/write head is used to read/write data onto tape. The main characteristic of tape is that data blocks can be accessed in sequential order. For this reason the access to data

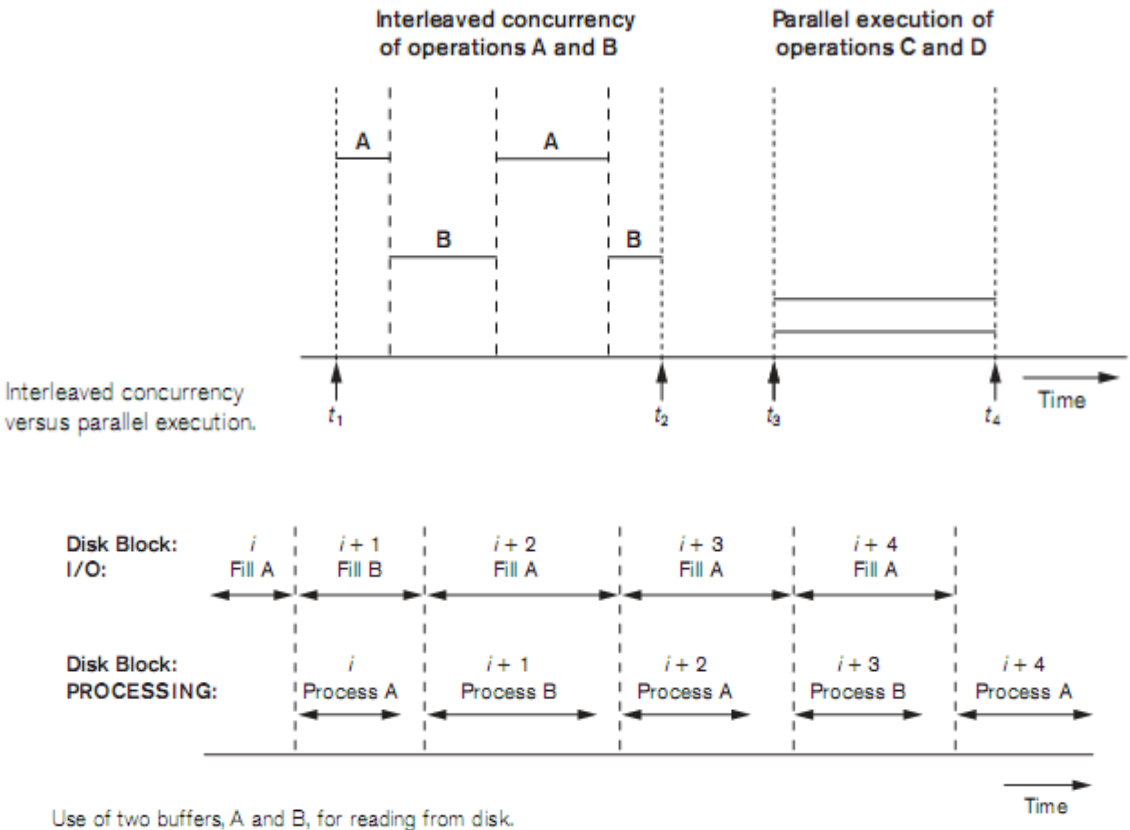
can be slow and hence are not used for online applications. Tapes are usually used for backing up the data base.

One reason for backup is to keep the copies of disk files in case the data is lost because of disk crash. Hence, disk files are periodically copied onto files. Database files that are seldom used or are outdated, but are required for historical record keeping can be archived on magnetic tapes.

7.3 Buffering of Blocks:

When several blocks need to be transferred from disk to main memory and all block addresses are known, several buffers can be reserved in the main memory to speed up the data transfer. While one buffer is read/written, the CPU can process data in the other buffer. This is possible because of an independent disk I/O controller that transfers data block from memory to hard disk independent of and in parallel to CPU processing. Buffering is most useful when processes can run concurrently in a parallel fashion either because separate disk I/O processor is available or multiple CPU processors exist.

Double Buffering: The CPU can start processing a block once its transfer to the main memory is completed. At the same time disk I/O processor can be reading and transferring the next block into a different buffer. This technique can be used to write a continuous stream of blocks from memory to disk. It permits continuous reading/writing of blocks thus, eliminating seek time and rotational delay for all except for the first block transfer.



7.4 Placing File Records on Disk

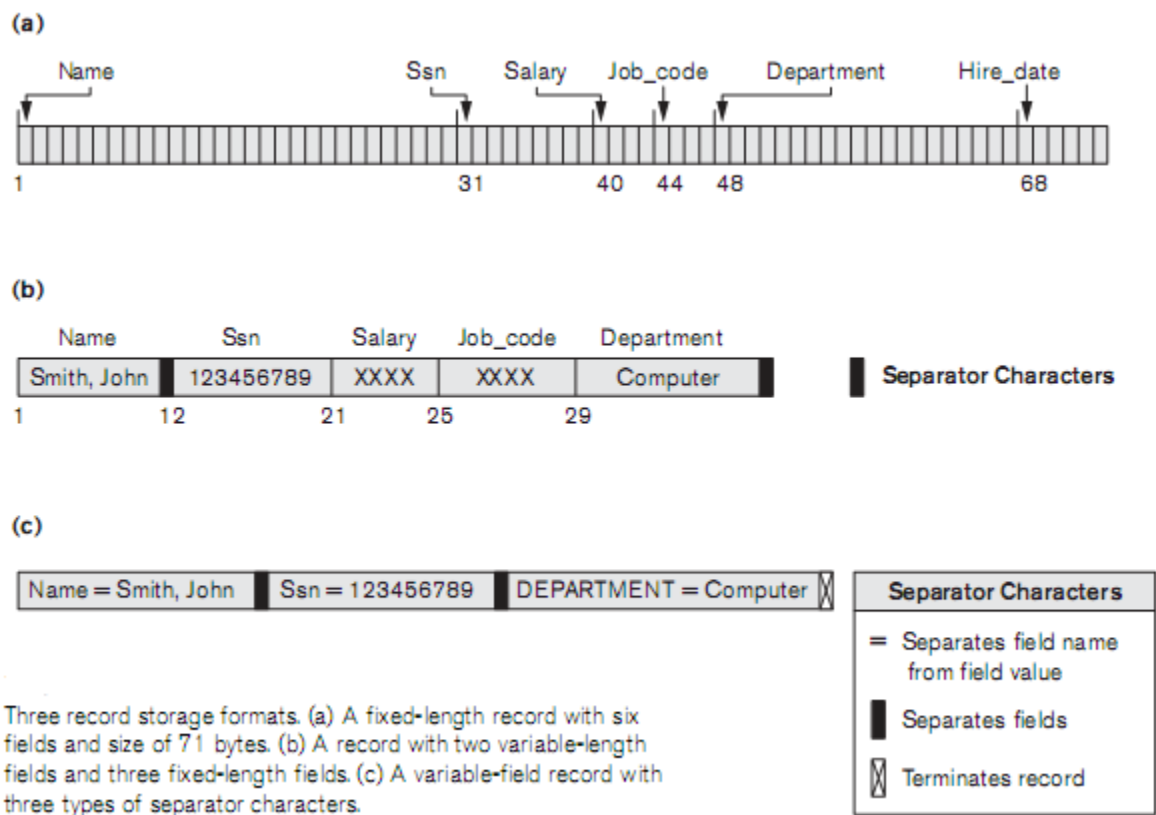
7.4.1 Files and Records:

Data are usually stored in the form of records. Each record is a collection of related data values. A collection of field names and their corresponding data type constitute a record type or record format definition. A file is a sequence of records. A file may contain 2 types of records.

- a. Fixed Length Records: If every record in the file has the same size, the file is said to be made of fixed length records. If different records in a file have different sizes, the file is said to be made up of variable length records.
- b. Variable Length Records: If different records in a file have different sizes then, the file is said to be made up of variable length records

A file may have variable length records because of the following reasons

- The file records are of same record type but one or more fields may be of varying size.
- The file records are of same record type but , one or more fields may have multiple value for the same field for individual records. Such fields are called repeating fields.
- A file may contain same record type but the fields may be optional.
- A file may contain records of different record types and hence of varying size.



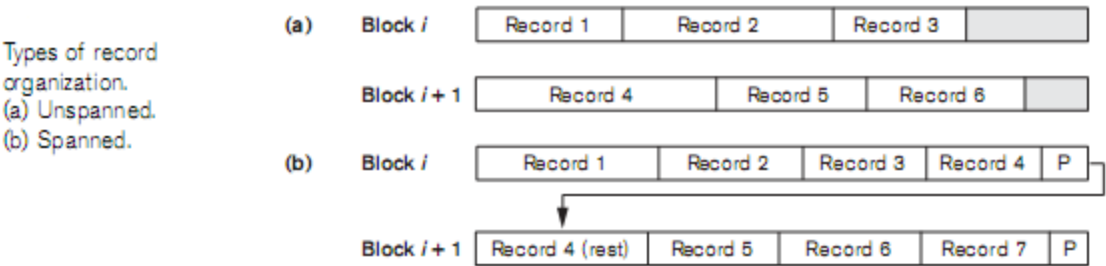
For variable-length fields, each record has a value for each field, but we do not know the exact length of some fields. To determine the bytes within a particular record that represent each field, we can use special characters (such as \$, %,?) - which do not appear in any field value to terminate variable length fields. These characters are called **separator characters**.

The values in the record are stored as <field name, field value> pairs. Separators are used to separate field name from field value, separate one field from the next and for repeating field. We can also assign a short **field type** code – say, an integer number to each field and include the record sequence as <field-type, field value>pairs. Repeating field needs one separator character to separate the repeating values of the field and another separator to indicate the termination of the field.

7.4.2 Record blocking:

Records in a file must be allocated to disk blocks because block is the unit of transfer between disk and memory. When the block size is larger than the record size, each block will contain numerous records. Suppose the block size is B bytes, for a file of fixed length records of size R bytes with $B \geq R$ we can fit

$bfr = \lfloor B/R \rfloor$ records per block. Here $\lfloor(x)\rfloor$ is the floor function that rounds down a number x to an integer. The value of bfr is called the blocking factor for the file. R may not divide B exactly, so we have some unused space in each block that is equal to $B - (bfr * R)$ bytes. To utilize this unused space, we can store a part of a record on one block and rest on another. A pointer at the end of first block points to a block that contains the rest of the record; in case the blocks are not consecutive. This organization is called **spanned** because records can span more than one block. When the record is larger than the block, we use spanned organization. Variable length records usually use spanned organization. When records are not allowed to cross block boundaries the organization is unspanned. This is generally used in fixed length records.



7.4.3 Allocating File Blocks on disks:

There are several techniques to allocate blocks of files onto disks. They are

- a. Contiguous allocation: In this type, the records are allocated consecutive blocks. Reading the file is fast and easy.
- b. Linked allocation: Each file contains a pointer to the next file block. In this type, it is easy to expand the file but reading the file becomes slow.

A combination of the above two approaches allocates clusters of consecutive disk blocks and the clusters are linked. Clusters are also called file segments or extents. Another method is indexed allocation, where one or more index blocks contain pointers to actual file blocks.

7.4.4 File Headers:

A **file header** or a **descriptor** contains information about a file that is needed by the system programs that access file records. It includes information to determine disk addresses of

file blocks, record format description which include field length, order of the fields and field type codes etc.

7.5 Operation on Files:

The two most common types of operations on files are

1. Retrieval operation – does not change any data but locates certain records to be processed by the user
2. Update operation- These operations make changes to a file by adding, deleting or modifying records.

In either case, we have to select one or more records based on the selection condition or filtering condition. The set of operations a DBMS software performs to process requests of users are

- Open – prepares the file for either reading or writing
- Reset – sets the file pointer of an open file to the beginning of a file
- Find/Locate – searches the first record that satisfies the search condition. Transfers the block containing that record into main memory buffer
- Read/Get – copies the current record from buffer to program variable in the user program
- FindNext – searches for the next record in the file that satisfies the search condition
- Delete – deletes the current record and updates the file on the disk to reflect deletion
- Modify – modifies the field values for current record and updates the file on the disk
- Insert – inserts a new record in the file by locating the block where the record is to be inserted
- Close – completes file access by releasing the buffers.

All the above operations except open/close are called record-at-time operations because they apply to single record.

The following are set at time operations that apply to a set of records

- Find All – locates all records that satisfy a given search condition
- Find n – searches for first record that satisfies a given search condition and then continues to locate next n-1 records
- Find ordered – retrieves all the records in a file in some specified order
- Reorganize – starts reorganizing process. Reorders the file records by sorting them in a specified order.

7.6 Files of unordered Records (Heap files):

This is the most basic type of organization where records are placed in a order in which they are inserted. New records are placed at the end of the file. Such files are known as heap files or pile files. It is used to collect and store records for future use. **Inserting** a new record is very efficient – the last disk block of file is copied into buffer ; the new record is added and the block is rewritten back to the disk. The address of the last file block is kept in the file header. **Searching** for a record using any search condition involves linear search through the file block by block. **Deleting** a record is also a slow process because we need to first locate the record to be

deleted. The program first find the a block, copies the block into the buffer, then delete the record from the buffer and finally rewrite the block back to the disk. This results in wastage of storage space. Another technique is to have an extra bit/byte for each record called the **deletion marker**. A record is deleted for a certain value in the deletion marker. A different value for the marker indicates that the record cannot be deleted. Both the above methods require periodic reorganization of the file to reclaim the unused space. This organization is also referred to as **sequential file organization**.

7.7 Files of Ordered Records (Sorted Files):

The records of a file can be physically ordered based on the values of one of their fields called the **ordering fields**. If the ordering field is also a key field, then it will have a unique value in each record. Such a field is called **ordering key** of the file. These files are known as **ordered sequential file**. Ordered records have the following advantages over unordered files.

1. Reading the records in the ordering of the key values becomes efficient because no sorting is required.
2. Finding a record in the ordering of the key field requires no additional block access.
3. A search condition based on the value of the ordering key results in faster access.

Binary search method is used to search records in sorted files. Inserting and deleting a record is expensive because the records are ordered. To insert a record, we must find a correct position in the file to insert the record in that position. In order to overcome the above situation two files are used. A transaction file – unordered file and a master file – ordered file. New records are added to the end of the transaction files. Periodically this file is sorted and merged with the master file during reorganization.

7.8 Hashing Techniques:

Hashing techniques provide very fast access to records on a certain condition. This organization is called a **hash file**. The search condition specified must be an equality condition on a single field called the **hash field** of the file. If the hash field is also a key field then it is called **hash key**. Hashing provides a function h called the hash function that is applied to the hash field value of the record and yields the address of the disk block in which the record is stored. A search for the record within the block can be carried out in a main memory buffer.

7.8.1 Internal Hashing:

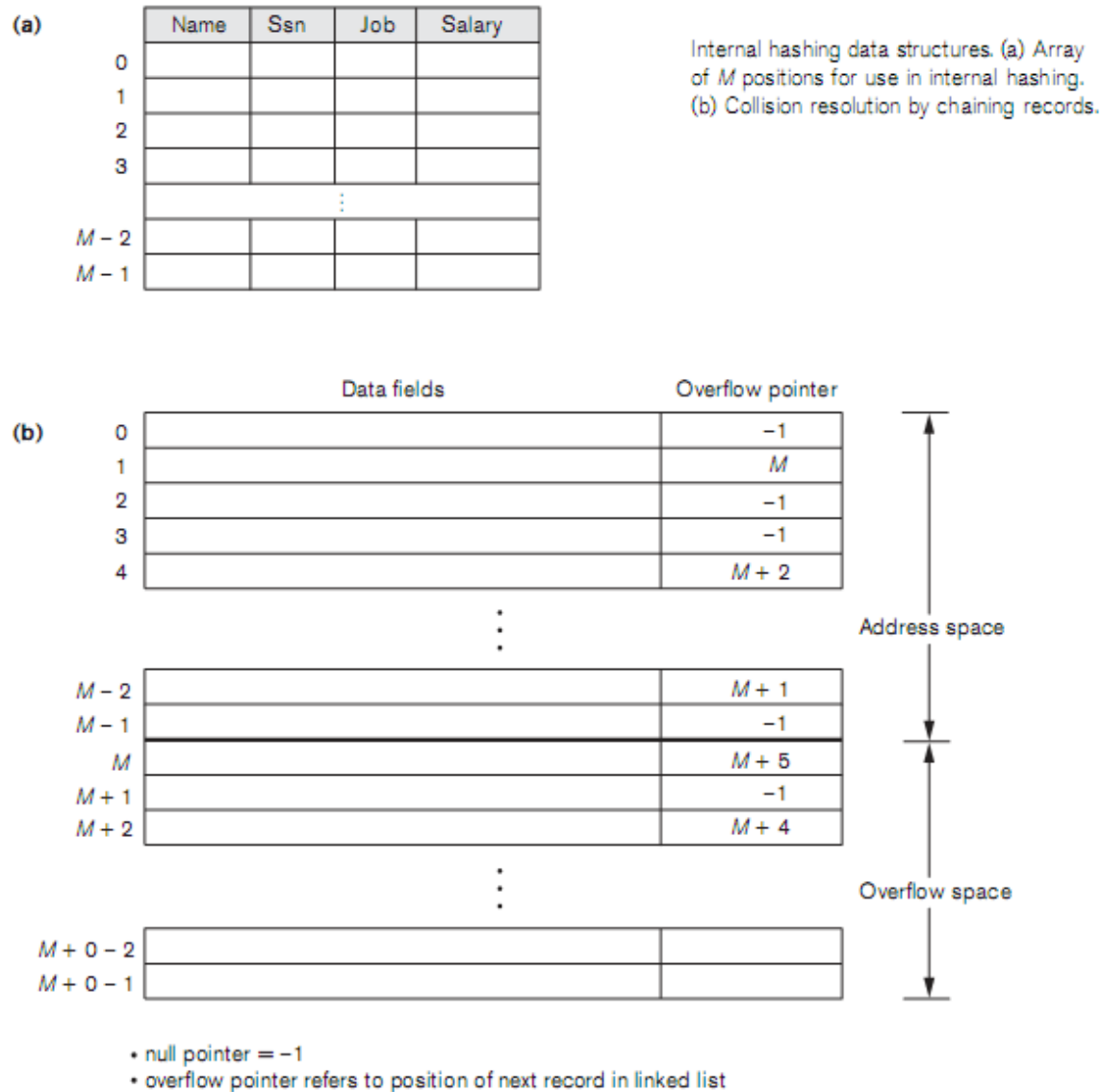
For internal files, hashing is typically implemented as a hash table through the use of array of records. Suppose the array index range is from 0 to $M-1$. We then have M slots whose addresses correspond to array indexes. We choose a hash function that transforms the hash field value into an integer between 0 and $M-1$. One most commonly used hash function is $h(K)=K \bmod M$ which returns a remainder of an integer hash field value K after division by M . This value is used for record address. Non integer hash field values can be transformed into integers before the mod function is applied. For characters strings, the numeric or ASCII value associated with the characters can be used in transformation.

One technique called folding involves applying an arithmetic function such as addition or a logical function such as exclusive OR to calculate hash address. The problem with most hashing functions is that they do not guarantee that distinct values will hash to distinct addresses,

because, the hash field space – the number of possible values a hash field can take is usually larger than the address space – the number of available addresses for records.

A **collision** occurs when the hash field value of the record that is being inserted hashes to an address that already contains a different record. In such situation, we must insert a new record in some other position, since its hash address is already occupied. The process of finding another position is called **collision resolution**. The different methods of collision resolution are:

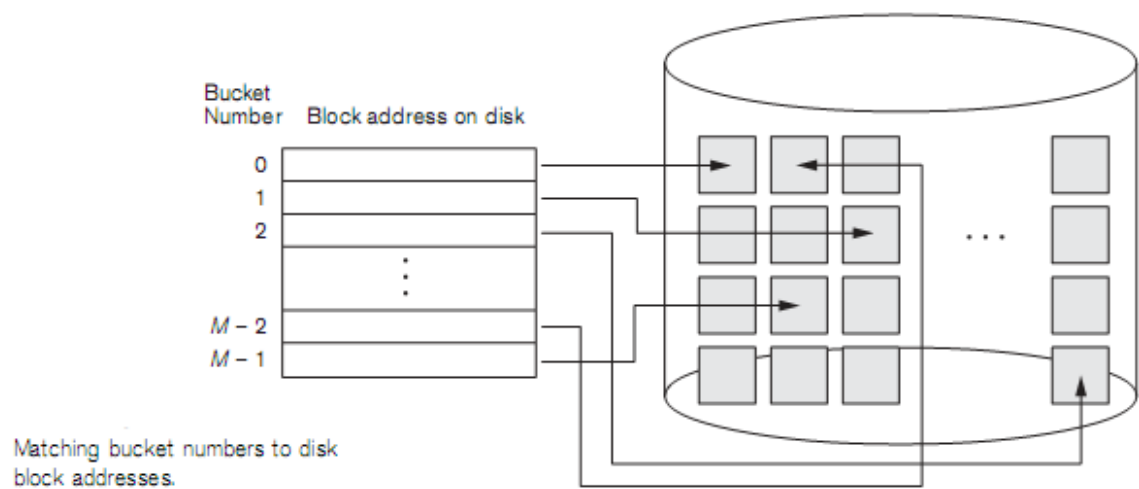
- Open addressing – Proceeding from the specified position specified by hash address, the program checks subsequent positions in order until an unused position is found.
- Chaining – Various overflow locations are kept, usually extending the array with the number of overflow positions. In addition a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location.
- Multiple hashing – The program applies a second hash function if the first results in a collision



7.8.2 External Hashing:

Hashing for disk files is called **external hash files**. To suit the characteristics of disk storage, the target address is made up of buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of contiguous blocks. The hash function maps a key into a relative bucket number, rather than assign an absolute block address to a bucket. A table maintained in the file header converts the bucket number into the corresponding disk block address.

The collision problem is less severe in with buckets, because there are many records that can fit in a single bucket. But, in case a collision occurs, it is resolved using record pointers. The pointers in the linked list include both the block address and the relative record position within the block. When a fixed number of buckets are allocated, the hashing scheme is called **static hashing**.



7.9 Other Primary Organizations:

7.9.1 Files of Mixed Records

The file organizations we have studied so far assume that all records of a particular file are of the same record type. The records could be of EMPLOYEEs, PROJECTs, STUDENTs, or DEPARTMENTs, but each file contains records of only one type. In most database applications, we encounter situations in which numerous types of entities are interrelated in various ways. Relationships among records in various files can be represented by **connecting fields**. For example, a STUDENT record can have a connecting field Major_dept whose value gives the name of the DEPARTMENT in which the student is majoring. This Major_dept field refers to a DEPARTMENT entity, which should be represented by a record of its own in the DEPARTMENT file. If we want to retrieve field values from two related records, we must retrieve one of the records first. Then we can use its connecting field value to retrieve the related record in the other file. Hence, relationships are implemented by logical field references among the records in distinct files.

7.10 RAID Technology:

RAID stands for Redundant Array of Independent Disks. The main goal of RAID is to improve the performance of disks like disk access time and disk capacities. Here the large arrays of small disks act as a single higher performance disk. A concept called data stripping is used, which utilizes parallelism to improve disk performance. Data stripping distributes data transparently over multiple disks to make them appear as a single large, fast disk. Stripping improves the overall I/O performance. It accomplishes load balancing among disks. Reliability can be improved by storing redundant information on disks.

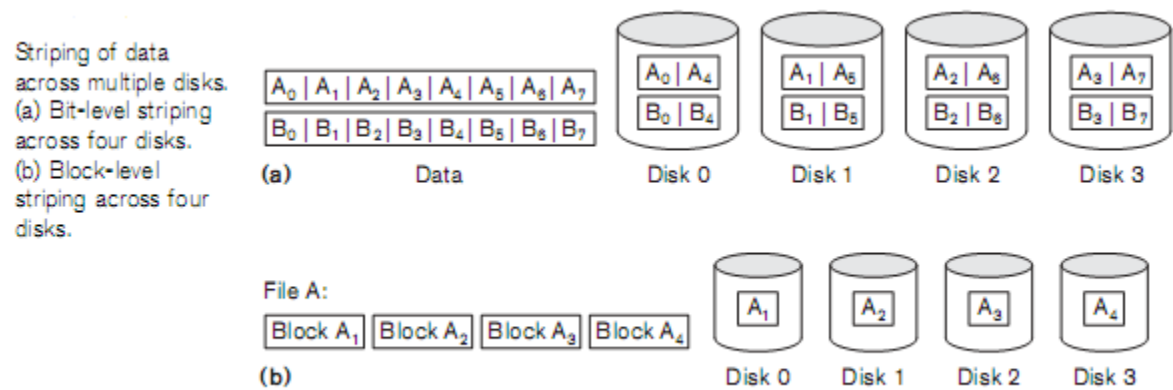
One technique for introducing redundancy is **mirroring /shadowing**. Data is written redundantly on two identical physical disks that are treated as one logical disk. When data is read, it can be retrieved from the disk with shorter seek time and rotational delay. If the disk fails another one is used until it is repaired. To incorporate redundancy, we must consider two problems

- 1. Selecting a technique for computing redundant information

2. Selecting methods of distributing redundant information across disk array.

The first problem can be solved by using error correcting codes involving parity bits or hamming codes. The second problem can be solved by storing redundant information on small number of disks or to distribute it uniformly across all disks.

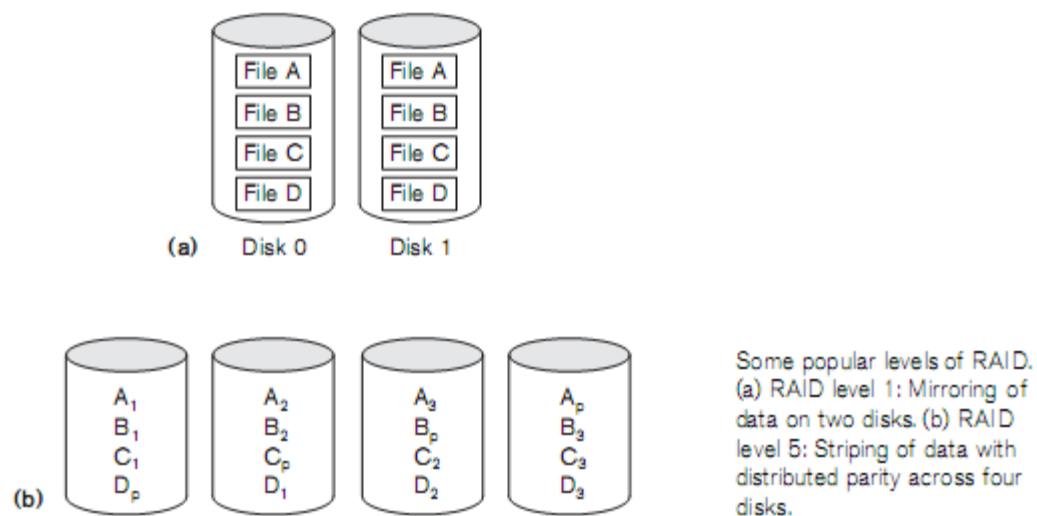
Data stripping can be applied at finer levels of granularity by breaking a byte of data into bits and spreading the bits across disks. This concept is called **bit-level data stripping** that consists of splitting a byte of data and writing bit j to the j^{th} disk. Even blocks of a file can be stripped across disks. This concept is called **block-level stripping**.



RAID Organization and Levels:

Different RAID organizations are defined based on data interleaving and pattern used to compute redundant information. The levels of RAID lie between 0 to 6.

RAID level 0 uses data striping and has no redundant data. It has got good write performance. RAID level 1 uses mirrored disks. It has good read performance. RAID level 2 uses memory style redundancy by using Hamming codes. It uses both error detection as well as error correction. RAID level 3 uses single parity disk relying on the disk controller to figure out which disk has failed. RAID levels 4 and 5 make use of block level stripping, with level 5 distributing data and parity information across all disks. RAID level 6 applies P+Q redundancy scheme using Reed-Solomon codes to protect against upto 2 disk failures by using just two redundant disks.



7.11 New Storage Systems:

Most large organizations have moved to a concept called storage area networks (SANs). In a SAN, online storage peripherals are configured as nodes on a high-speed network and can be attached and detached from servers in a very flexible manner. Several companies have emerged as SAN providers and supply their own proprietary topologies. They allow storage systems to be placed at longer distances from the servers and provide different performance and connectivity options. The main advantages claimed include:

- Flexible many-to-many connectivity among servers and storage devices using fiber channel hubs and switches
- Up to 10 km separation between a server and a storage system using appropriate fiber optic cables
- Better isolation capabilities allowing non-disruptive addition of new peripherals and servers

Assignment

Short Answers (2 marks)

1. Define mirroring
2. Define stripping
3. Define seek time
4. Define block transfer time
5. What are the limitations in internal hashing?
6. What do you mean by track?
7. Define cylinder.

Long answers (4 or more marks)

1. What are the reasons for variable length records? What types of separator characters are needed for each?
2. Write a note on magnetic tape storage device.
3. Explain the function of disk controller in a magnetic disk.

4. What are sorted files? What are the advantages of such a file over unordered files?
5. Explain different operations on file.
6. Discuss hash file organization with reference to disk files.
7. Explain about RAID technology.
8. Explain the concept of double buffering. How it improves data access?
9. Explain Heap (unordered) file organization. Also mention the drawbacks.
10. Explain the hardware mechanism of hard disk.
11. Explain the concept of internal hashing.
12. Explain the concept of external hashing.

UNIT III

CHAPTER 8

STRUCTURED QUERY LANGUAGE

8.1 Oracle and client-server Technology:

8.1.1 Oracle server

- Oracle server is a multi-user tool that works in a client/server environment. Client/server Programming is a distributed application program.
- It has three distinct components, each focusing on a specific job. The three components are
 1. Oracle Server
 2. Oracle Client Tool
 3. Network for connecting the first two component
- Oracle server's primary job is to manage data optimally, among multi-users that concurrently request for the same data.
- Oracle client tools are that part of the system that provides an interface to the user so that the data retrieved from the server can be manipulated.
- It allows users to pass data manipulation requests to the oracle server.
- The network and communicating software is the vehicle that transports data between oracle client tools and the oracle server.

8.1.2 Oracle Client side tools are:

1. Oracle SQL*Plus
2. Oracle Form Designer
3. Oracle Reports Designer
4. Oracle Graphics

Oracle SQL*Plus

- Oracle server is a separate tool that comes as a part of oracle enterprise server as well as Oracle Workgroup server via which user can communicate interactively with the Oracle server.
- Oracle provide an interactive SQL tool called SQL*Plus, which allow user to enter ANSI SQL sentences and pass them to the Oracle Engine for execution.
- To use ANSI SQL the user must load the SQL*Plus tool in clients memory, link to the server & then communicate with the Oracle Engine loaded on the server.

8.2 Data Manipulation in DBMS

SQL commands are instructions used to communicate with the database to perform specific task that work with data. SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users. SQL commands are grouped into four major categories depending on their functionality:

- Data Definition Language (DDL) - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- Data Manipulation Language (DML) - These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.
- Transaction Control Language (TCL) - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- Data Control Language (DCL) - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

Features of SQL

- SQL can be used by a range of users, including those with little or no programming experience.
- It is a non procedural language.
- It reduces the amount of time required for creating and maintaining systems.
- It is an English-like language.

8.3 Data types in Oracle

Oracle supports the following data types.

- char
- varchar2
- number
- Date

8.3.1 CHAR

The CHAR datatype stores fixed-length character strings. When you create a table with a CHAR column, you must specify a string length (in bytes or characters) between 1 and 2000 bytes for the CHAR column width. The default is 1 byte. Oracle then guarantees that:

When you insert or update a row in the table, the value for the CHAR column has the fixed length.

- If you give a shorter value, then the value is blank-padded to the fixed length.
- If a value is too large, Oracle Database returns an error.

8.3.2 VARCHAR2

The VARCHAR2 data type stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column. For each row, Oracle Database stores each value in the column as a variable-length field unless a value exceeds the column's maximum length, in which case Oracle Database returns an error. Using VARCHAR2 and VARCHAR saves on space used by the table.

For example, assume you declare a column VARCHAR2 with a maximum size of 50 characters. In a single-byte character set, if only 10 characters are given for the VARCHAR2 column value in a particular row, the column in the row's row piece stores only the 10 characters

8.3.3 NUMBER

The NUMBER datatype stores fixed and floating-point numbers. Numbers of virtually any magnitude can be stored and are guaranteed portable among different systems operating Oracle Database, up to 38 digits of precision. For numeric columns, you can specify the column as:

```
column_name NUMBER
```

Optionally, you can also specify a precision (total number of digits) and scale (number of digits to the right of the decimal point):

```
column_name NUMBER (precision, scale)
```

If a precision is not specified, the column stores values as given. If no scale is specified, the scale is zero.

8.3.4 DATE

The DATE datatype stores point-in-time values (dates and times) in a table. The DATE datatype stores the year (including the century), the month, the day, the hours, the minutes, and the seconds (after midnight). For input and output of dates, the standard Oracle date format is DD-MON-YY e.g. 13-AUG-89

8.4 SQL Commands

8.4.1 CREATE

Purpose:

The CREATE TABLE statement is used to create a new table in the database. Tables are defined in part by the columns they contain. Each column has a specific data type which specifies how data is stored in the column. When creating a new table, you must decide on the appropriate data type of the column. These data types are then specified in the CREATE TABLE Statement.

Syntax:

a. The general format of CREATE command is

```
CREATE TABLE tablename  
(column1 data type(size),  
 column2 data type(size),  
 column3 data type(size),  
 :  
 column-n data type(size));
```

b. We can create a table from another table in SQL. The syntax is as follows

```
CREATE TABLE <new table> (<column1>, <column2>)  
AS SELECT <column1>, <column2> FROM <existing table>;
```

Example:

a. Create a table called student that contains roll number, name and marks in three subjects

```
SQL> create table student
```

```
2 (rollno number(2),
3 name varchar2(20),
4 m1 number(2),
5 m2 number(2),
6 m3 number(2));
```

Table created.

b. Create a table named sample from the existing table student that contains only rollno and names of students

```
SQL> create table sample(rollno, name)
```

```
2 as select rollno,name from student;
```

Table created.

c. Create a table named sample1 that contains the same structure as student but with no records

```
SQL> create table sample1
```

```
2 as select * from student where 1=2;
```

Table created.

```
SQL> select * from sample1;
```

no rows selected

8.4.2 INSERT**Purpose:**

The INSERT statement is used to insert records into a table. This statement loads the table with data to be manipulated later. When inserting a single row into the table the insert operation does the following:

- Creates a new row in the database table
- Loads the values passed into the columns specified

Note: Character data type values must be enclosed in single quotes(').

Syntax:

```
INSERT INTO <table name> (column1, column2,....., column n)
VALUES(<expression1>, <expression2>,.....<expression n>);
```

Example:

```
SQL> insert into student(rollno,name,m1,m2,m3) values (11,'Rama',65,75,50);
```

1 row created.

```
SQL> insert into student values(&rollno,&'name',&m1,&m2,&m3);
```

Enter value for rollno: 12

Enter value for name: Raju

Enter value for m1: 70

Enter value for m2: 80

Enter value for m3: 65
old 1: insert into student values(&rollno,&name',&m1,&m2,&m3)
new 1: insert into student values(12,'Raju',70,80,65)
1 row created.

8.4.3 SELECT

Purpose:

Once data is inserted into the tables, the next logical operation would be to view the data that has been inserted. The SELECT verb in SQL is used to achieve this. The SELECT command is used to retrieve rows from one or more table

Syntax:

- a. To select all rows and all columns
SELECT * FROM <table name>;
- b. To retrieve selected columns and all rows
SELECT column1, column2 FROM <table name>;
- c. To retrieve selected rows and all columns
SELECT * FROM <table name> WHERE <condition>;
- d. To retrieve selected columns and selected rows
SELECT column1, column2 FROM <table name> WHERE <condition>;
- e. To eliminate duplicate rows when using select statement – The **DISTINCT** clause in the select statement allows removing of duplicate rows from the result set. The DISTINCT clause can only be used with select statements. It scans through the values in the columns specified and displays only unique values amongst them.
SELECT DISTINCT column1, column2 FROM <table name>;

Example:

- a. Display the details of all the students
SQL> select * from student;

ROLLNO NAME		M1	M2	M3
11	Raja	50	60	70
12	Rama	70	80	75
13	John	45	55	65
14	Gita	40	50	45
15	Tom	30	35	25

5 rows selected

- b. Display the roll number and names of all the students
SQL> select rollno,name from student;

```
ROLLNO NAME
-----
11 Raja
12 Rama
13 John
14 Gita
15 Tom
5 rows selected
```

- c. Display the details of students who got more than 60 in subject 2
SQL> select * from student where m2>=60;

```
ROLLNO NAME          M1    M2    M3
-----
11 Raja             50    60    70
12 Rama             70    80    75
2 rows selected
```

- d. Display names of students who secured more than 50 either in subject1 or subject 3
SQL> select name from student where m1>=50 or m3>=50;

```
NAME
-----
Raja
Rama
John
3 rows selected
```

- e. Display the different courses available
SQL> select distinct course from student;

```
COURS
-----
bca
bbm
mca
3 rows selected
```

8.4.4 DELETE

Purpose:

The DELETE command is used to delete rows from a table that satisfies the condition and returns the number of records that were deleted. If the DELETE command is executed without a WHERE clause then, all rows are deleted. This verb either deletes all the rows or a set of rows in a given table

Syntax:

- a. DELETE FROM <table name>;
- b. DELETE FROM <table name> WHERE <condition>;

Example:

- a. Remove all the records from sample table.

```
SQL> select * from sample;
```

```
ROLLNO NAME
```

```
-----
```

```
11 Raja
```

```
12 Rama
```

```
13 John
```

```
14 Gita
```

```
15 Tom
```

```
5 rows selected
```

```
SQL> delete from sample;
```

```
5 rows deleted.
```

- b. Remove the details of student 'Raja'

```
SQL> delete from student where name='Raja';
```

```
1 row deleted.
```

8.4.5 ALTER**Purpose:**

The structure of a table can be modified by using the ALTER TABLE command. ALTER TABLE allows the user to change the structure of the existing table. With ALTER TABLE command it is possible to

- Add or delete the columns
- Create or destroy indexes
- Change the data type of existing columns
- Rename columns or table itself

ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, then original table is deleted and the new one is modified.

Syntax:

- a. Adding new tables

```
ALTER TABLE <table name>
```

```
ADD(<new column> <data type>(size),
```

```
:
```

```
<new column> <data type>(size));
```

- b. Modifying existing columns

```
ALTER TABLE <table name>
```

```
MODIFY(<column> <new datatype>(new-size));
```

Example:

- a. Add a new column Total with data type number and size 2

```
SQL> alter table student
2 add (total number(2));
Table altered.
```

- b. Change the size of total to 3

```
SQL> alter table student
2 modify (total number(3));
Table altered.
```

8.4.6 UPDATE

Purpose:

The UPDATE command is used to change or modify data values in a table. The UPDATE verb in SQL is used to either update all the rows or selected rows in a table. The UPDATE statement updates columns in the existing table's rows with new values. The SET clause indicates which column data should be modified and the new values they should hold. The WHERE clause, if given, specifies which rows should be updated. Otherwise all the rows are updated.

Syntax:

- a. UPDATE <table name>
SET <column 1>=<expression>,
:
<column n>=<expression>;
- b. UPDATE <table name>
SET <column 1>=<expression>,
:
<column n>=<expression>
WHERE <condition>;

Example:

- a. Increment the marks of all students by 10 in subject2
SQL> update student set m2=m2+10;
4 rows updated.
- b. Update the total of all the students
SQL> update student
2 set total=m1+m2+m3;
4 rows updated.
- c. Update the student details by changing the name of Rama to Sriram
SQL> update student
2 set name='Sriram'
3 where name='Rama';

1 row updated.

8.4.7 RENAME

Purpose:

RENAME command is used to rename or give another name to the existing table.

Syntax:

RENAME <old table name> TO <new table name>;

Example:

SQL> rename student to student1;
Table renamed.

8.4.8 DESCRIBE

Purpose:

The DESCRIBE command is used to display the structure of a table. This command displays the column names, the data types and the special attributes that are connected to the table.

Syntax:

DESCRIBE <table name>;

Example:

SQL> describe student;

Name	Null?	Type

ROLLNO		NUMBER(2)
NAME		VARCHAR2(20)
M1		NUMBER(2)
M2		NUMBER(2)
M3		NUMBER(2)
TOTAL		NUMBER(3)

8.4.9 DROP

Purpose:

The DROP TABLE statement is used to destroy a specific table. If a table is dropped all the records held within it are lost can cannot be recovered.

Syntax:

DROP TABLE <table name>;

Example:

SQL> drop table student1;
Table dropped.

8.5 Computations on table data:

8.5.1 Arithmetic Operators

Oracle allows arithmetic operators to be used while viewing records from a table or while performing data manipulation operations. The various arithmetic operators are as follows

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

8.5.2 Comparison Operators

Comparison operators are used to compare the column data with specific values in a condition. Comparison Operators are also used along with the SELECT statement to filter data based on specific conditions.

The below table describes each comparison operator.

Operators	Description
=	equal to
<>, !=	is not equal to
<	Less than
>	Greater than
<=	less than or equal to
>=	greater than or equal to

Examples:

- a. List the roll number and names student whose total is more than 200
SQL> select rollno,name from student
2 where total>200;

ROLLNO	NAME
12	Sriram
17	Shyam

2 rows selected

- b. List the names of students who secured less than 50 in subject1
SQL> select name from student
2 where m1<50;

NAME
John
Gita
Tom

3 rows selected

- c. List the names of students who got more than or equal to 70 in subject3
SQL> select name from student
2 where m3>=70;

NAME
Sriram
Shyam

2 rows selected

8.5.3 Logical operators

There are three Logical Operators namely, AND, OR, and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output. Logical operators are used in the WHERE clause and allows you to combine more than one condition.

Logical Operators	Description
OR	For the row to be selected at least one of the conditions must be true.
AND	For a row to be selected all the specified conditions must be true.
NOT	For a row to be selected the specified condition must be false.

Example:

- a. List the details of students who secured more than 60 in subject 1 and subject 2
SQL> select * from student where m1>60 and m2>60;

ROLLNO	NAME	M1	M2	M3	TOTAL	COURSE
12	Sriram	70	90	75	235	bca

1 row selected

- b. List the names of students who secured less than 50 in any subject.
SQL> select name from student where m1<50 or m2<50 or m3<50;

NAME
John
Gita
Tom

3 rows selected

8.5.4 Range Searching

In order to select data that is within a range of values, the BETWEEN operator is used. The BETWEEN operator allows the selection of rows that contain values within a specified lower and upper limit. The two values in between the range must be linked with a keyword AND. The BETWEEN operator can be used with both character and numeric data types.

Example:

- a. List the names of students whose total is in the range 180 and 250

SQL> select * from student where total between 180 and 250;

ROLLNO	NAME	M1	M2	M3	TOTAL	COURSE
12	Sriram	70	90	75	235	bca
17	Shyam	80	60	70	210	bbm

2 rows selected

8.5.5 Pattern Matching

LIKE Operator

The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose we use a the following wildcard character

- % allows to match any string of any length
- _ allows to match on a single character.

Example:

- a. List the names of students whose name begins with ‘R’

SQL> select name from student where name like 'S%'

NAME

Sriram

Sita

Shyam

3 rows selected

- b. List the name and roll number of students whose name ends with ‘a’

SQL> select rollno,name from student where name like '%a';

ROLLNO NAME

14 Gita

16 Sita

2 rows selected

- c. List the names of students who have ‘i’ as the second letter in their name.

SQL> select name from student where name like '_i%';

NAME

Gita

Sita

2 rows selected

- d. List the course whose name ends with the letter ‘a’

SQL>select course from student where course like '%m'

COURSE

bbm
bbm
2 rows selected

IN and NOT IN

The arithmetic operator (=) compares a single value to another single value. In case a value needs to be compared to a list of values then the IN predicate is used. It reduces the need to use multiple OR conditions in a query.

Example:

- a. List the student names and course who are studying either mca or bca
SQL> select name, course from student where course in ('mca','bca');

NAME	COURSE
Sriram	bca
Gita	bca
Sita	mca
Tom	Mca

4 rows selected

- b. List the student names along with their course who are not studying mca
SQL> select name, course from student where course not in ('mca')

NAME	COURSE
Sriram	Bca
John	Bbm
Gita	bca
Shyam	Bbm

4 rows selected

8.6 DUAL and SYSDATE

Oracle table – DUAL

Dual is a table owned by SYS. SYS owns the data dictionary and DUAL is a part of data dictionary. Dual is a small Oracle work table which consists of only one row and one column. It contains the value x in that column. It is used to perform small arithmetic calculations and also supports date retrieval and formatting.

SQL> describe dual;

Name	Null?	Type

DUMMY		VARCHAR2(1)

SQL> select * from dual;

DUAL
X

Example:

SQL> select 3*5 from dual;
3*5

15

SYSDATE is a pseudo column that contains the current date and time. It requires no arguments when selected from the table DUAL and returns the current date.

Example:

```
SQL> select sysdate from dual;
```

```
SYSDATE
```

```
-----
```

```
18-NOV-15
```

8.7 Oracle Functions

Oracle functions serve the purpose of manipulating data items and returning the result. Functions are also capable of accepting user-supplied variables or values and operating on them. Such variables are called arguments. The general format of a function is
Function_name(argument1, argument2,...)

Oracle functions can be classified depending on whether they operate on a single row or a group of rows. Oracle functions can be classified as

- Group Functions or Aggregate Functions
- Scalar functions or single row functions

8.7.1 Group Functions (Aggregate Functions)

Functions that act on a set of values are called group functions. A group function returns a single result row for a group of queried rows. A list of group functions with examples is given below.

Examples:

- **AVG** – Returns the average of the values in a column
E.g. Display the average total of all the students

```
SQL> select avg(total) "Average" from student;
```

```
Average
```

```
-----
```

```
171.666667
```
- **MAX** – Returns the maximum value in a given column
E.g. Display the maximum total from student table

```
SQL> select max(total) "Maximum Total" from student;
```

```
Maximum Total
```

```
-----
```

```
235
```
- **MIN** – Returns the minimum value in a given column
E.g. Display the minimum total from student table

```
SQL> select min(total) "Minimum Total" from student;
```

```
Minimum Total
```

```
-----
```

```
100
```
- **COUNT** – Returns the number of rows in a table

E.g. Display the number of students in student table
SQL> select count(*) from student;

COUNT(*)

6

- SUM – Returns the sum of values in a given column
E.g. Display the sum total of marks in subject1 from student table
SQL> select sum(m1) "Marks in Subject1" from student;
Marks in Subject1

295

8.7.2 Scalar Functions (Single row Functions)

Functions that act only on one value at a time are called scalar functions. A single row function returns one result for every row of a queried table. They can further be grouped depending on the data type of value upon which they act. The classification of scalar functions is as follows:

- String functions – String data type
- Numeric functions – Numeric data type
- Date functions – Date data type
- Conversion functions – converting one data type to another

String Functions:

String functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output. Few of the character or text functions are as given below:

Function Name	Return Value
LOWER (string_value)	All the letters in 'string_value' is converted to lowercase.
UPPER (string_value)	All the letters in 'string_value' is converted to uppercase.
INITCAP (string_value)	All the letters in 'string_value' is converted to mixed case.
LTRIM (string_value, trim_text)	All occurrences of 'trim_text' is removed from the left of 'string_value'.
RTRIM (string_value, trim_text)	All occurrences of 'trim_text' is removed from the right of 'string_value' .
TRIM (trim_text FROM string_value)	All occurrences of 'trim_text' from the left and right of 'string_value' , 'trim_text' can also be only one character long .
SUBSTR (string_value, m, n)	Returns 'n' number of characters from 'string_value' starting from the 'm' position.
LENGTH (string_value)	Number of characters in 'string_value' in

	returned.
LPAD (string_value, n, pad_value)	Returns 'string_value' left-padded with 'pad_value' . The length of the whole string will be of 'n' characters.
RPAD (string_value, n, pad_value)	Returns 'string_value' right-padded with 'pad_value' . The length of the whole string will be of 'n' characters.

Examples:

Function Name	Examples
LOWER(string)	SQL> select lower('Srinivas') from dual <u>LOWER('S</u> Srinivas
UPPER(string)	SQL> select upper('Srinivas') from dual; <u>UPPER('S</u> SRINIVAS
INITCAP(string)	SQL> select initcap('sri nivas') from dual; <u>INITCAP('</u> Sri Nivas
LTRIM(string, trim_text)	SQL> select ltrim('nivas','ni') from dual; <u>LTR</u> Vas
RTRIM(string, trim_text)	SQL>select rtrim('sreenivas','nivas') from dual <u>RTRI</u> Sree
SUBSTR(string_value, start, number of characters)	SQL> select substr('Srinivas',4,5) from dual; <u>SUBST</u> Nivas
LENGTH (string_value)	SQL> select length('Srinivas') from dual; <u>LENGTH('SRINIVAS')</u> 8
LPAD (string_value, n, pad_value)	SQL> select lpad('Nivas',10, '*') from dual; <u>LPAD('NIVA</u> *****Nivas
RPAD (string_value, n, pad_value)	SQL> select rpad('Nivas',10, '*') from dual <u>RPAD('NIVA</u> Nivas*****

Numeric Functions:

Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:

Function Name	Return Value
---------------	--------------

ABS (x)	Absolute value of the number 'x'
CEIL (x)	Integer value that is Greater than or equal to the number 'x'
FLOOR (x)	Integer value that is Less than or equal to the number 'x'
TRUNC (x, y)	Truncates value of number 'x' up to 'y' decimal places
ROUND (x, y)	Rounded off value of the number 'x' up to the number 'y' decimal places
POWER(m,n)	Returns m raised to n th power. Here n must be an integer

The following examples explains the usage of the above numeric functions

Function Name	Examples
ABS (x)	SQL> select abs(-26) from dual; <u>ABS(-26)</u> 26
CEIL (x)	SQL> select ceil(34.6) from dual; <u>CEIL(34.6)</u> 35
FLOOR (x)	SQL> select floor(34.6) from dual; <u>FLOOR(34.6)</u> 34
TRUNC (x, y)	SQL> select trunc(38.546,2) from dual; <u>TRUNC(38.546,2)</u> 38.54
ROUND (x, y)	SQL> select round(38.546,2) from dual <u>ROUND(38.546,2)</u> 38.55
POWER(m,n)	SQL> select power(5,2) from dual; <u>POWER(5,2)</u> 25

Date Functions:

These functions are used to manipulate and extract values from the date column of a table

Function Name	Return Value
ADD_MONTHS(date,n)	Returns the date after adding the number of months specified in the function
LAST_DAY(date)	Returns the last date of the month specified in the function
MONTHS_BETWEEN(d1,d2)	Returns the number of months between d1 and d2

Examples of the above said functions:

Function Name	Example
ADD_MONTHS(date,n)	SQL>select add_months('25-sep-2013',3) from dual <u>ADD_MONTH</u> 25-DEC-13
LAST_DAY(date)	SQL> select last_day('25-sep-2013') from dual; <u>LAST_DAY(</u> 30-SEP-13
MONTHS_BETWEEN(d1,d2)	SQL>select months_between('25-sep-2013','25-nov-2013') from dual; <u>MONTHS BETWEEN('25-SEP-2013','25-NOV-2013')</u> -2 SQL>select months_between('25-sep-2013','25-apr-2013') from dual; <u>MONTHS BETWEEN('25-SEP-2013','25-APR-2013')</u> 5

Conversion Functions:

These are functions that help us to convert a value in one form to another form. For example, a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE.

Few of the conversion functions available in oracle are:

Function Name	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by 'date_format'.
NVL (x, y)	If 'x' is NULL, replace it with 'y'. 'x' and 'y' must be of the same datatype.

The below table provides the examples for the above functions

Function Name	Examples
TO_CHAR ()	SQL> select to_char(sysdate,'dd-month-yyyy') from dual; <u>TO_CHAR(SYSDATE,'</u> 25-september-2013
TO_DATE ()	SQL> select to_date('15-aug-1947','dd-month-yy') from dual; <u>TO_DATE('</u> 15-AUG-47

8.8 Data Constraints on a table:

There are two types of data constraints that can be applied to data being inserted into an oracle table. One type of constraint is the I/O constraint that determines the speed at which data can be inserted or extracted from a table. The other type of constraint is called the business rule constraint.

8.8.1 I/O constraints

The various types of I/O constraints are

- Primary key constraint – A primary key is one or more columns in a table use to uniquely identify rows in a table. None of the fields that are a part of primary key can contain null values. A table can have only one primary key. The data in that column should be unique i.e. no duplicate values are allowed. The column is a mandatory column i.e. it cannot be left blank and NOT NULL attribute
- Foreign key constraint – Foreign keys represent relationship between tables. A foreign key is a column whose values are derived from primary key of some other tables. The table in which the foreign key is defined is called **foreign table**. The table in which the primary key is defined is called the **primary table or master table**. The master table can be referenced in the foreign key definition by using the **references** clause.

Oracle displays an error message when the record in the master table is deleted and corresponding records exists in the foreign table. It prevents the delete operation to be performed. For this reason, Oracle provides **ON DELETE CASCADE** option. If the **ON DELETE CASCADE** option is set, a delete operation in the master table will trigger a DELETE operation for all the corresponding records in all the foreign tables.

- Unique key constraint

8.8.2 Business rule constraints

Oracle allows programmers to define constraints at table level and column level. If the constraints are defined as an attribute of a column definition when creating or altering the table structure, they are **column level constraints**. If data constraints are defined after defining all the table column attributes when creating or altering a table structure, then it is table level constraints. The various business rule constraints are

- NOT NULL – A NULL is different from zero or blank. A NULL value can be inserted into columns of any data type. The NOT NULL constraint ensures that a table column cannot be left empty. When a column is defined as **NOT NULL**, then that column becomes mandatory. It implies that a value must be entered into the column if that record is to be accepted for storage in the table.

Syntax:

<column name> <data type(size)> NOT NULL

- CHECK constraint – This constraint must be specified as a logical expression that evaluates either to a TRUE or FALSE. E.g. Data values being inserted to begin with a particular letter, gender to accept only M or F, data values to be accepted only in uppercase.

Syntax:

a. Check constraint at Column level:

<column name> <data type(size)> check(logical expression)

b. Check constraint at Column level:
check(logical expression)

Example:
Create a table EMPLOYEE using SQL command to store details of employees such as EMPNO, NAME, DESIGNATION, DEPARTMENT, GENDER and SALARY. Specify Primary Key and NOT NULL constraints on the table and allow only ‘M’ or ‘F’ for the column GENDER. Write the following SQL queries :

```
SQL> create table EMPLOYEE
2 (empno varchar2(4) PRIMARY KEY,
3  name varchar2(20) NOT NULL,
4  designation varchar2(20) NOT NULL,
5  dept varchar2(20),
6  gender char(1),
7  salary number(6) NOT NULL,
8  check (gender in('M','F')));
```

Table created.

8.9 User Constraints Table

A table can be created with multiple constraints attached to its columns. The user can view the structure along with its constraints using the DESCRIBE command. This command only displays the column names, data type, size and not null constraints. The information about other constraints like primary key, foreign key is not available using the describe command. Oracle stores such information in a table called the USER CONSTRAINTS. Some of the columns available in USER CONSTRAINTS table is listed below.

Column Name	Description
OWNER	The owner of the constraint
CONSTRAINT_NAME	The name of the constraint
TABLE_NAME	The name of the table associated with the constraint
CONSTRAINT_TYPE	The type of constraint P:Primary key R:Foreign key U:Unique key C:Check constraint
SEARCH_CONDITION	Search condition used in check constraint
R_OWNER	The owner of table referenced by foreign key
R_CONSTRAINT_NAME	The name of the constraint referenced by foreign key

8.10 Defining and dropping integrity constraints in ALTER table:

Integrity constraints can be applied to tables using the ALTER table command.
Example:

a. Make Rollno field as primary key in student table
SQL> alter table student add primary key(rollno);
Table altered.

b. Show that the above constraint has been applied using USER CONSTRAINTS table
SQL>select owner, table_name, constraint_type from user_constraint where
table_name='STUDENT';

OWNER	TABLE_NAME	C
SCOTT	STUDENT	P

Integrity constraints can be dropped if the constraint is no longer needed. This can be achieved by the ALTER table using the DROP clause.

Example:

- a. Drop primary key constraint from student
SQL> alter table student drop primary key;
Table altered.

8.11 Default value Concept:

Default values can be assigned to columns when we create a table. When a record is loaded into the table and a column is left empty, oracle engine will automatically load this column with the default value specified. The data type of the default value must match with the data type specified for the column.

Syntax:

<column name> <data type(size)> default <value>;

8.12 GROUP BY and HAVING Clause:

The GROUP BY clause tells Oracle to group rows based on distinct values that exists from specified columns. The group by clause creates a data set containing several sets of records grouped together based on a condition.

Syntax:

SELECT <column1>,<column2>....<column n>
AGGREGATE FUNCTION(Expression)
FROM table name WHERE <condition>
GROUP BY <column1>,<column2>....<column n>;
HAVING

Example:

- a. Display the number of students in each course
SQL> select course, count(*) "Number of Students" from student
2 group by course;

COURS Number of Students

```
bca      2
bbm      2
mca      2
3 rows selected
```

- b. Display the maximum total from each course
SQL> select course, max(total) from student
2 group by course;

```
COURS MAX(TOTAL)
-----
bca      235
bbm      210
mca      165
3 rows selected
```

The HAVING clause is used in conjunction with the group by clause. It imposes a condition on the group by clause, which further filters the groups created by the group by clause. Each column specification specified in the having clause must occur in the list of columns mentioned in the group by clause.

Example:

- a. List the course with more than 1 student.
SQL> select course from student
2 group by course
3 having count(*)>1;

```
COURSE
bca
mca
```

8.13 ORDER BY Clause:

The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order. Oracle sorts query results in ascending order by default.

Syntax:

```
SELECT column-list
FROM table_name [WHERE condition]
[ORDER BY column1 [, column2, .. columnN] [DESC]];
```

Example:

- a. Display the student name and total marks in ascending order.
SQL> select name,total from student
2 order by total;

```
NAME      TOTAL
```

Tom	100
Gita	145
Sita	165
Shyam	210
Sriram	235
5 rows selected	

b. Display the student names and marks in subject 1 in descending order
SQL> select name,m1 from student
2 order by m1 desc;

NAME	M1

Sriram	70
Shyam	60
Sita	50
Gita	40
Tom	30
5 rows selected	

8.14 Sub Queries

A subquery is a form of SQL statement that appears inside another SQL statement. It is also called a nested query. The statement containing the subquery is called parent query. The parent statement uses the result set returned by the subquery. It can be used for

- Inserting records in the target table
- Create and insert records in a table
- Update records in target table
- To provide values for conditions in the WHERE, HAVING, IN clause used with SELECT, UPDATE and DELETE commands

The concept of using a subquery in the FROM clause of the SELECT statement is called **inline view**.

A **correlated subquery** is one where a subquery references a column from a table in the parent query. A correlated subquery is evaluated once for each row of the parent statement, which can be any of the SELECT, UPDATE or DELETE.

The **EXISTS** operator is usually used with correlated subqueries. This operator enables to test whether a value retrieved by the outer query exists in the result set of the values retrieved by the inner query. If the subquery returns at least one row, the operator returns **true**. If the value does not exist, it returns **false**. The **EXISTS** operator ensures that the search in the inner query terminates when at least one match is found. The **NOT EXISTS** operator enables to test whether a value retrieved by the outer query is not a part of the result set of the values retrieved by the inner query.

Examples:
Consider the information in the following tables
SQL> select * from employ;

EMPNO	ENAME	SAL	DEPTNO
E001	Raja	6000	D002
E002	Rani	7000	D001
E003	Jaya	5000	D003
E004	Ranbir	8000	D001
E005	Rajni	4000	D004
E006	Sita	5500	D003

6 rows selected.

SQL> select * from depart;

DNO	DNAME	LOC
D001	accounts	chennai
D002	research	mumbai
D003	HRD	bangalore
D004	EDP	hyderabad
D005	Inventory	Goa

4 rows selected

- a. Display the names of employees who are working in Chennai

SQL> select ename from employ

2 where deptno in (select dno from depart where loc='chennai');

ENAME

Rani

Ranbir

2 rows selected

- b. Display the location where Sita is working

SQL> select loc from depart

2 where dno=(select deptno from employ where ename='Sita');

LOC

Bangalore

- c. List the employee number, employee name, salary and average salary of the department whose salary is more than the average salary in the department(inline view)

SQL> select a.empno,a.ename,a.sal,b.avgсал

2 from employ a,(select deptno,avg(сал) avgсал from employ group by deptno) b

3 where a.deptno=b.deptno and a.sal>b.avgсал

EMPNO ENAME SAL AVGSAL

E006 Sita 5500 5250

E004 Ranbir 8000 7500

- d. List the employee names, salary who are drawing maximum salary in each department (correlated subquery)

SQL> select ename,deptno,sal from employ a

2 where сал = (select max(сал) from employ where deptno=a.deptno);

ENAME	DEPT	SAL
-----	-----	-----
Raja	D002	6000
Ranbir	D001	8000
Rajni	D004	4000
Sita	D003	5500

- e. List the departments that do not have any employees
SQL> select dno,dname from depart a
2 where not exists(select deptno from employ where deptno=a.dno);

DNO	DNAME
-----	-----
D005	inventory

- f. List the employee names who are working in accounts department
SQL> select ename from employ a
2 where exists (select dname from depart where dno=a.deptno and dname='accounts');

ENAME

Ranbir
Rani

8.15 Joins

There are occasions where we need to retrieve data from multiple tables. This is achieved in SQL using joins. Tables are joined on columns that have same data type and data width in the tables. The tables are related with the help of primary key and foreign keys of the table. The different types of joins are

- INNER JOIN
- OUTER JOIN
- CROSS JOIN
- SELF JOIN

INNER JOIN – These joins are also called equi joins. They are known as equi joins because the where clause compares two columns using the = operator. It is the most commonly used operator. It returns all the rows from both the tables where there is a match.

Example:

- a. Display the names of employees and the department in which they work
SQL> select ename,dname from employ, depart
2 where employ.deptno=depart.dno;

ENAME	DNAME
-----	-----
Raja	research
Rani	accounts
Jaya	HRD

```
Ranbir    accounts
Rajni     EDP
Sita      HRD
6 rows selected.
```

- b. Display the names of employees working in HRD department
- ```
SQL> select ename from employ, depart
2 where employ.deptno=depart.dno
3 and dname='HRD';
```

```
ENAME

Jaya
Sita
```

**OUTER JOIN** – This type of join can be used in selecting rows from both the tables regardless of whether the tables have common values or not. NULL values are appended in the result set where the data is missing. For left outer join use a (+) to the left condition and for right outer join use a (+) to the right condition.

**Syntax for Left outer join**

```
SELECT table1.column, table2.column
FROM table1 t1, table2 t2
WHERE t1.column(+) = t2.column;
```

**Syntax for Right outer join**

```
SELECT table1.column, table2.column
FROM table1 t1, table2 t2
WHERE t1.column = t2.column(+);
```

Example:

- a. Display the employee name and department using left outer join
- ```
SQL> select ename, dname
2  from employ e, depart d
3  where e.deptno(+) = d.dno;
```

```
ENAME      DNAME
-----
Raja       research
Rani       accounts
Jaya       HRD
Ranbir     accounts
Rajni      EDP
Sita       HRD
           inventory
7 rows selected.
```

- b. Display the employee name and department using right outer join
- ```
SQL> select ename, dname
```

```
2 from employ e, depart d
3 where e.deptno=d.dno(+);
```

| ENAME            | DNAME    |
|------------------|----------|
| -----            |          |
| Ranbir           | accounts |
| Rani             | accounts |
| Raja             | research |
| Sita             | HRD      |
| Jaya             | HRD      |
| Rajni            | EDP      |
| Gita             |          |
| 7 rows selected. |          |

CROSS JOIN – A cross join returns the Cartesian product. This means that the join combines every row from the left table with every row in the right table.

Example:

```
SQL> select * from sam1;
```

| ENO   | ENAME   |
|-------|---------|
| ----- |         |
| E001  | janani  |
| E002  | jahnavi |

```
SQL> select * from sam2;
```

| PNO   | PNAME   |
|-------|---------|
| ----- |         |
| P001  | soap    |
| P002  | shampoo |

- a. Illustrate a cross join operation on sam1 and sam2
- ```
SQL> select eno,ename,pno,pname
2 from sam1 cross join sam2;
```

ENO	ENAME	PNO	PNAME

E001	janani	P001	soap
E001	janani	P002	shampoo
E002	jahnavi	P001	soap
E002	jahnavi	P002	shampoo

SELF JOIN – A self-join is one where the same table is involved in the join operation

Example:

- a. Display the employee names and salary who earn more than Sita
- ```
SQL> select e1.ename,e1.sal
2 from employ e1, employ e2
```

3\* where (e1.sal>e2.sal) and (e2.ename='Sita');

| ENAME  | SAL   |
|--------|-------|
| -----  | ----- |
| Raja   | 6000  |
| Rani   | 7000  |
| Ranbir | 8000  |

8.16 UNION, INTERSECT and MINUS

Union Clause:

Multiple queries can be put together and their output can be combined using the union clause. The union clause merges the output of two or more queries into a single set of rows. Here, the number of columns and the data type of the columns must be the same in all the select statements used in the query.

Intersect Clause:

Multiple queries can be put together and their output can be combined using the intersect clause. It outputs only the rows produced by both the queries intersected. The output in the intersect clause will include only those rows that are retrieved common to both the queries. Here, the number of columns and the data type of the columns must be the same in all the select statements used in the query.

Minus Clause:

Multiple queries can be put together and their output can be combined using the minus clause. The minus clause outputs the rows produced by the first query, after filtering the rows retrieved by the second query. Here, the number of columns and the data type of the columns must be the same in all the select statements used in the query.

8.17 Security Management in SQL

Oracle provides extensive security features in order to safeguard the information stored in its tables from unauthorized access. The rights that allow the use of some or all the Oracle's resources on the server are called **privileges**. Objects that are created by the user are owned and controlled by that user. If a user wishes to access any of the objects belonging to another user, the owner of the object will have to give permissions for such access. This is called **Granting of Privileges**. Privileges once given can be taken back by the owner of the object. This is called **Revoking of privileges**.

Granting Privileges using the GRANT statement:

The grant statement provides various types of access to database objects.

Syntax:

GRANT <object privileges>  
ON <object name>  
TO <user name>;

Revoking Privileges:

Privileges once given can be denied to the user using the REVOKE command. The object owner can revoke privileges granted to another user.

**Syntax:**

GRANT <object privileges>  
ON <object name>  
FROM <user name>;

**Assignment****Short Answers (2 marks)**

1. Give syntax and example for the following commands in SQL:  
CREATE, INSERT, SELECT, ALTER, UPDATE, RENAME, DROP
2. Give the syntax use of functions INTCAP(), POWER(), ROUND().
3. Explain any four data types that a table in Oracle can hold.
4. How to sort a table in increasing and decreasing order? Explain with example.
5. What are the two levels of constraints? Give examples.
6. Write a note on pattern matching using % and \_ in the LIKE clause.
7. What is sub query? Explain with example.
8. How do you define foreign key at column level and table level?
9. List any four aggregate functions.
10. What do you mean by correlated sub query? Give example.
11. What is the purpose of check constraint in SQL?
12. What do you mean by privileges?
13. Give the syntax of GRANT command

**Long answers(4 or more marks)**

1. What is join? How do you join table with itself? Explain with explicit cursor attributes.
2. Explain the following with examples.  
i) COUNT(\*) ii) BETWEEN iii) UPDATE
3. Explain SELECT statement with example.
4. Explain range searching and pattern matching predicates with examples.
5. Explain i) GROUP BY ii) ORDER BY clause.
6. What are nested queries? With an example each, explain the nested queries using EXISTS and NOT EXISTS clause.
7. What do you mean by correlated subquery? Explain with an example.
8. Explain any five string functions in Oracle with examples.
9. Explain any five numeric functions in Oracle with examples.
10. Explain any four aggregate functions.

UNIT IV  
CHAPTER 9  
Introduction to PL/SQL

Introduction:

SQL suffers from inherent disadvantages. They are

- It does not have procedural capabilities
- SQL statements are passed to the oracle engine one at a time. This decreases the processing speed in a multi-user environment
- SQL has no facility of programmed handling of error messages

9.1 Advantages of PL/SQL:

- PL/SQL is a development tool that not only supports SQL data manipulation but also provides facilities for condition checking, branching and looping
- PL/SQL sends the entire block of SQL code to oracle engine. As the entire block of SQL code is passed to Oracle engine at one time for execution, all the changes made to the data in the table are done or undone in one go.
- PL/SQL facilitates the displaying of user-friendly messages when errors are encountered
- PL/SQL allows declaration and use of variables in blocks of code. The variables are used to store intermediate results of a query for later processing
- PL/SQL all sorts of calculations can be done efficiently and quickly
- Applications written in PL/SQL are portable to any computer hardware system and operating system

9.2 Generic PL/SQL block

A single PL/SQL code block consists of a set of SQL statements, clubbed together, and passed to oracle engine entirely. This block has to be logically grouped together for the engine to recognize it as singular block of code. A PL/SQL block has a definite structure, which can be divided into sections. The sections of PL/SQL block are

- a. Declare section
- b. Master begin...end section

|           |                                                                                      |
|-----------|--------------------------------------------------------------------------------------|
| DECLARE   | Declarations of memory variables, constants, cursors                                 |
| BEGIN     | SQL executable statements<br>PL/ SQL executable statements                           |
| EXCEPTION | Handles errors that arise during execution of code block between BEGIN and EXCEPTION |
| END;      | End of PL/SQL block                                                                  |

The Declare Section:

Code block starts with the declaration section, in which memory variables and other oracle objects are declared, and initialized if necessary. Once declared they can be used in SQL statements for data manipulation.

**The Begin Section:**

It consists of a set of SQL and PL/SQL statements which describe processes to be applied to table data. Actual data manipulation, retrieval, branching and looping constructs are specified in this section.

**The Exception Section:**

This section deals with handling of errors that arise during the execution of data manipulation statement, which make up the PL/SQL code block. Errors can arise due to syntax, logic or validation rule violation

**The End Section:**

This marks the end of a PL/SQL block.

**Structure of PL/SQL block:**

```
DECLARE
 <declarations section>
BEGIN
 <executable command(s)>
EXCEPTION
 <exception handling>
END;
```

**Example:**

```
DECLARE
 message varchar2(20):= 'Hello, World!';
BEGIN
 dbms_output.put_line(message);
END;
/
```

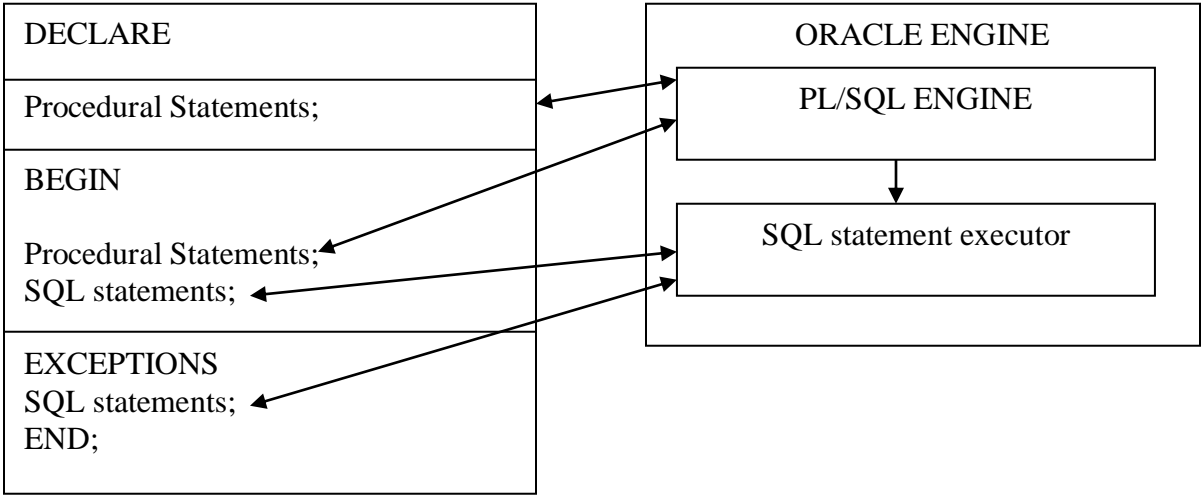
Output:

Hello, World!

**9.3 PL/SQL Execution Environment:**

- The PL/SQL engine accepts any valid PL/SQL block as input
- PL/SQL engine resides in the Oracle engine. Oracle engine can process both SQL statements and PL/SQL blocks
- These blocks are sent to the PL/SQL engine, where the procedural statements are executed. SQL statements are sent to the SQL executor in the Oracle engine.
- The call to the oracle engine needs to be made only once to execute any number of SQL statements if these SQL statements are bundled inside a PL/SQL block





9.4 PL/SQL Character set:

The basic character set in PL/SQL includes

- Upper case letters {A..Z}
- Lower case letters {a..z}
- Numerals {0..9}
- Symbols ( ) + - \* / < > = ! @ %

Words used in PL/SQL blocks are called **lexical units**. Blank spaces can freely be inserted between lexical units in a PL/SQL block. A **literal** is a numeric or character string used to represent itself. A **numeric literal** can be a integer or float value. E.g. 5, -20, 29.6. A **string literal** is a set of two or more characters enclosed in single quotes. E.g. ‘Hello’, ‘Hi’. A **character literal** is a single character enclosed in single quotes. E.g. ‘Y’, ‘N’, ‘m’. **Boolean literals or logical literals** are predetermined constants. The values these literals can hold are TRUE, FALSE, NULL.

9.5 PL/SQL Data types:

The default data types that can be declared in PL/SQL are

- Number – Numeric values, on which arithmetic operations are performed.
- Character – Alphanumeric values that represent single characters or strings of characters.
- Boolean – Logical values, on which logical operations are performed.
- Date/time – Date and time.

The above data types can have NULL values. The % type attribute is used to declare variables based on the definitions of columns in a table. It helps to declare a variable or constant to have same data type as that of a column declared in a table.

Variables:

A variable in PL/SQL is a named variable. A variable name must begin with a character and can be followed by a maximum of 29 characters. Reserved words cannot be used as variable

names unless enclosed within double quotes. A blank space cannot be embedded in a variable name. variable names are not case sensitive. Variables are separated from each other by a comma. Values can be assigned to variables in two ways:

1. Using the assignment operator :=
2. Selecting or fetching table data values into variables

**Constants:**

A constant is declared in the same way as how a variable is declared. The only difference is that we use a keyword constant and a value is assigned to it immediately. Thereafter no further assignments can be made.

**Logical Comparisons:**

Comparisons in PL/SQL can be made using Boolean expressions. A Boolean expression consists of variables separated by relational operators{<,>,<=,>=,<>}. A logical expression is a combination of two or more Boolean expressions separated by logical operators{AND,OR, NOT}. A boolean expression always evaluates to TRUE, FALSE or NULL

**Displaying user messages on the VDU:**

DBMS\_OUTPUT is a package that includes a number of functions and procedures that accumulate information in a buffer so that it can be retrieved later. These functions can also be used to display a user message on the screen. PUT\_LINE puts a piece of information in package buffer followed by end of the line marker. It expects a single parameter of character data type. In order to display a message, SERVEROUTPUT should be set ON. It is a SQL\*Plus environment parameter that displays the information passed as parameter to the PUT\_LINE function.

**Syntax:**

SET SERVEROUTPUT [ON/OFF]

**Comments:**

A comment in PL/SQL can be written in two ways:

1. It can begin with a double hyphen(--). Here, the entire line can be treated as a comment.
2. A group of lines can be enclosed between /\* and \*/. All the lines will be ignored for processing

Example:

- a. --Single line comment
- b. /\* This is  
a comment\*/

**Example PL/SQL code:**

```
SET SERVEROUTPUT ON
DECLARE
 -- constant declaration
 pi constant number := 3.141592654;
 -- other declarations
 radius number(5,2);
 circumference number(7, 2);
```

```
area number (10, 2);
BEGIN
 -- processing
 radius := 9.5;
 circumference := 2.0 * pi * radius;
 area := pi * radius * radius;
 -- output
 dbms_output.put_line('Radius: ' || radius);
 dbms_output.put_line('Circumference: ' || circumference);
 dbms_output.put_line('Area: ' || area);
END;
SET SERVEROUTPUT OFF
/
Output:

Radius: 9.5
Circumference: 59.69
Area: 283.53
```

### 9.6 Control Structures:

The flow of control statements can be classified as

- Conditional control
- Iterative control
- Sequential control

#### 9.6.1 Conditional control:

PL/SQL allows the use of an IF statement to control the execution of a block of code. In PL/SQL, IF-THEN-ELSE structure allows to check certain conditions under which a specific block of code should be executed.

It is the simplest form of IF control statement, frequently used in decision making and changing the control flow of the program execution. The IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF. If the condition is TRUE, the statements get executed and if the condition is FALSE or NULL then the IF statement does nothing.

The different forms of IF statements are as follows:

1) IF condition THEN  
statement 1;  
END IF;

2) IF condition THEN  
statement 1;  
ELSE  
statement 2;  
END IF;

```
3)IF condition 1 THEN
 statement 1;
 statement 2;
ELSIF condition2 THEN
 statement 3;
ELSE
 statement 4;
END IF;
```

**Example:**

```
SQL> ed samif.sql
set serveroutput on
declare
 x number(2);
 y number(2);
begin
 x:=&x;
 y:=&y;
 if x>y then
 dbms_output.put_line (x || 'is bigger');
 else
 dbms_output.put_line (y || 'is bigger');
 end if;
end;
```

**Output:**

```
Enter value for x: 35
old 5: x:=&x;
new 5: x:=35;
Enter value for y: 25
old 6: y:=&y;
new 6: y:=25;
35is bigger
PL/SQL procedure successfully completed.
```

**9.6.2 Iterative control:**

Iterative control indicates the ability to repeat sections of code. A loop marks the sequence of statements that has to be repeated. Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.

There are three types of loops in PL/SQL:

- Simple Loop
- While loop
- For loop

**Simple Loop:**

A Simple Loop is used when a set of statements is to be executed at least once before the loop terminates. An EXIT condition must be specified in the loop, otherwise the loop will get into an infinite number of iterations. When the EXIT condition is satisfied the process exits from the loop.

**Syntax of Simple Loop:****LOOP**

statements;

[EXIT WHEN condition;]

**END LOOP;**

These are the important steps to be followed while using Simple Loop.

1. Initialize a variable before the loop body.
2. Increment the variable in the loop.
3. Use a EXIT WHEN statement to exit from the Loop. If you use a EXIT statement without WHEN condition, the statements in the loop is executed only once

Example:

set serveroutput on

declare

i number:=0;

begin

dbms\_output.put\_line('Even numbers from 1 to 10 are:');

loop

i:= i+2;

dbms\_output.put\_line (i);

exit when i&gt;=10;

end loop;

end;

Output:

Even numbers from 1 to 10 are:

2

4

6

8

10

PL/SQL procedure successfully completed.

**While loop:**

A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true. The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.

Syntax of WHILE LOOP is:

hoWHILE &lt;condition&gt;

LOOP statements;

END LOOP;

Example:

```
set serveroutput on
declare
 pi constant number(5,2):=3.14;
 r number(2);
 c number(5,2);
begin
 r:=3;
 while r<=7
 loop
 c:=2*pi*r;
 insert into circle values(r,c);
 r:=r+1;
 end loop;
end;
```

Output:

SQL> @samwhile.sql;  
PL/SQL procedure successfully completed.

SQL> select \* from circle;

| RADIUS | CIRCUM |
|--------|--------|
| -----  | -----  |
| 3      | 18.84  |
| 4      | 25.12  |
| 5      | 31.4   |
| 6      | 37.68  |
| 7      | 43.96  |

**FOR Loop**

A FOR LOOP is used to execute a set of statements for a predetermined number of times. Iteration occurs between the start and end integer values given. The counter is always incremented by 1. The loop exits when the counter reaches the value of the end integer.

Syntax of FOR LOOP is:  
FOR counter IN value1..value2  
  LOOP statements;  
END LOOP;

Example:

set serveroutput on

```
declare
 i number;
 n number;
 f number:=1;
begin
 n:=&n;
 for i in 1..n
 loop
 f:=f*i;
 end loop;
 dbms_output.put_line('Factorial is '|| f);
end;
```

Output:

Enter value for n: 5

old 6: n:=&n;

new 6: n:=5;

Factorial is 120

PL/SQL procedure successfully completed.

### 9.6.3 Sequential Control

The GOTO statement changes the flow of control within a PL/SQL block. This statement allows execution of a section of code which is not in the normal flow of control. The entry point to such a block of code is marked using the tags << user-defined name>>. The GOTO statement can then make use of this label to jump to that block of code for execution.

Syntax:

GOTO <code-block name>

### 9.7 Oracle Transactions

A series of one or more SQL statements that are logically related or a series of operations performed on the Oracle table data is called as a **transaction**. Oracle treats changes made to data in a table as a two step process. First, the requested changes are done. In order to make these changes permanent, a **COMMIT** statement is issued at the SQL prompt. A **ROLLBACK** statement issued at the SQL prompt can be used to undo a part of or the entire transaction. A transaction begins with the first executable statement after a commit, rollback or a connection made to the Oracle engine. A transaction is a group of events that occur between any of the following events:

- Connecting to Oracle
- Disconnecting from Oracle
- Committing changes to database table
- Rollback

**Closing Transactions** – A transaction can be closed using either a commit or rollback statement. By using these statements, table data can be changes or all the changes made to the table data can be undone

**COMMIT** – A commit ends the current transaction and makes permanent any changes made to the table data during a transaction.

**Syntax:**

commit;

**SAVEPOINT** – It marks and saves the current point in the processing of a transaction. When a savepoint is used with a rollback statement, parts of the transaction can be undone. An active savepoint is one that is specified since the last COMMIT or ROLLBACK.

**Syntax:**

savepoint <savepointname>;

**ROLLBACK** – A rollback does exactly the opposite of COMMIT. It ends the transaction but undoes any changes made during the transaction. Rollback can be fired at the SQL prompt with or without savepoint clause.

**Syntax:**

rollback [work] [to [savepoint]<savepointname>];

### Assignment

#### Short Answers (2 marks)

1. What are the limitations of SQL?
2. How do you write comments in PL/SQL?
3. What is the purpose of GOTO statement in PL/SQL?
4. Give the syntax of for loop in PL/SQL?
5. Give the syntax of simple if in PL/SQL?
6. Give the syntax of if-else in PL/SQL?
7. What is an Oracle transaction?
8. What is the purpose of commit in PL/SQL?
9. What is the purpose of rollback in PL/SQL?
10. What is the purpose of savepoint in PL/SQL?

#### Long answers(4 or more marks)

1. Explain the advantages of PL/SQL.
2. Give syntax of IF and explain with example.
3. Explain PL/SQL block structure with example.
4. Explain if-else statement in PL/SQL with an example.
5. Explain loop statement in PL/SQL with an example.
6. Explain while statement in PL/SQL with an example.
7. Explain for statement in PL/SQL with an example.



CHAPTER 10  
CURSORS

10.1 Introduction:

The Oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is private to SQL operations and is called a **Cursor**. The data that is stored in the cursor is called the **Active Data Set**. The size of the cursor in the memory is the size required to hold the number of rows in the active data set. Oracle has a predefined area in the main memory set aside, within which cursors are opened. . A cursor contains information on a select statement and the rows of data accessed by it. A cursor can hold more than one row, but can process only one row at a time

Cursors are classified depending on the circumstances under which they are opened. Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed

10.2 Implicit cursors

If the oracle engine opened a cursor for its internal processing, it is known as **implicit cursor**. These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

Implicit Cursor Attributes:

| Attributes | Return Value                                                                                                                                                     | Example      |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| %FOUND     | The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ....INTO statement return at least one row. | SQL%FOUND    |
|            | The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT....INTO statement do not return a row.               |              |
| %NOTFOUND  | The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECT ....INTO statement return at least one row.           | SQL%NOTFOUND |
|            | The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE do not affect even one row and if SELECT ....INTO statement does not return a row.   |              |
| %ROWCOUNT  | Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE,                                                                                 | SQL%ROWCOUNT |

|  |        |  |
|--|--------|--|
|  | SELECT |  |
|--|--------|--|

Example: Consider the PL/SQL Block that uses implicit cursor attributes as shown below:

```
/*increase the salary of all employees by 1000*/
DECLARE var_rows number(5);
BEGIN
 UPDATE employ
 SET salary = salary + 1000;
 IF SQL%NOTFOUND THEN
 dbms_output.put_line('None of the salaries where updated');
 ELSIF SQL%FOUND THEN
 var_rows := SQL%ROWCOUNT;
 dbms_output.put_line('Salaries for ' || var_rows || 'employees are updated');
 END IF;
END;
```

**Output:**  
Salaries for 7employees are updated  
PL/SQL procedure successfully completed.

10.3 Explicit cursors

A cursor can be opened They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row. When individual records in a table have to be processed inside a PL/SQL block explicit cursor is used. This cursor will be declared and mapped to an SQL query in the declare section of the PL/SQL block and used with the executable section of PL/SQL block.

The steps involved in using explicit cursor and manipulating data are

- Declare a cursor in the declare section of PL/SQL block
- Open the cursor
- Fetch the data from the cursor one row at a time into the memory variables
- Process the data that was fetched
- Exit from the loop after the processing is complete
- Close the cursor

Explicit cursor attributes:

| Attributes | Return Value                                                                                                                                                     | Example            |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| %ISOPEN    | Evaluates to TRUE, if an explicit cursor is open. It returns FALSE if the cursor is closed                                                                       | Cursor_name%ISOPEN |
| %FOUND     | The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ....INTO statement return at least one row. | Cursor_name%FOUND  |

|           |                                                                                                                                                                |                      |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
|           | The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT....INTO statement do not return a row.             |                      |
| %NOTFOUND | The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECT ....INTO statement return at least one row.         | Cursor_name%NOTFOUND |
|           | The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE do not affect even one row and if SELECT ....INTO statement does not return a row. |                      |
| %ROWCOUNT | Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT                                                                        | Cursor_name%ROWCOUNT |

**Functionality of Open, Fetch and Close commands:**

**Declaring a cursor** – A cursor is declared in the declare section of the PL/SQL block. This is done to create an active data set. The cursor name is used to reference the active data set that is held within the cursor

**Syntax:**

CURSOR cursor-name IS SELECT statement;

**Example:**

```
DECLARE
 CURSOR emp_cur IS
 SELECT *
 FROM employ
 WHERE salary > 5000;
```

In the above example we are creating a cursor ‘emp\_cur’ on a query which returns the records of all the employees with salary greater than 5000.

**Opening a cursor** – Opening a cursor executes a query and creates an active data set that contains all rows which meet the search criteria of the query. An open statement retrieves the records from the table and places the records in the cursor

**Syntax:**

OPEN cursor\_name;

**Fetching records from the cursor** – The fetch statement retrieves the rows from the active data set into the memory variables declared in the PL/SQL block one row at a time. Each time a fetch statement is executed, the cursor pointer is advances to the next row in the active data set.

**Syntax:**

FETCH cursor-name INTO variable list;

**Closing a cursor** – The close statement disables the cursor and the active data set becomes undefined. This will release the memory occupied by the cursor and the active data set.

**Syntax:**

```
CLOSE cursor-name;
```

General Form of using an explicit cursor is:

```
DECLARE
 variables;
 records;
 create a cursor;
BEGIN
 OPEN cursor;
 FETCH cursor;
 process the records;
 CLOSE cursor;
END;
```

Example:

```
/*Display employee name and salary whose salary is more than 6000*/
```

```
SET SERVEROUTPUT ON
```

```
DECLARE
 emp_rec employ%rowtype;
 CURSOR emp_cur IS
 SELECT *
 FROM employ
 WHERE sal > 6000;
BEGIN
 OPEN emp_cur;
 FETCH emp_cur INTO emp_rec;
 dbms_output.put_line (emp_rec.ename || ' ' || emp_rec.sal);
 CLOSE emp_cur;
END;
```

```
/
```

Output:

```
SQL> @EXPCUR.SQL
```

```
Gita 7000
```

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

#### 10.4 Cursor FOR LOOP:

When using FOR LOOP you need not declare a record or variables to store the cursor values, need not open, fetch and close the cursor. These functions are accomplished by the FOR LOOP automatically.

**General Syntax for using FOR LOOP:**

```
FOR record_name IN cursor_name
LOOP
```

```
 process the row...
END LOOP;
```

Example:

```
/*To display all employees whose salary is greater than 9000*/
SET SERVEROUTPUT ON
DECLARE
 CURSOR emp_cur IS
 SELECT *
 FROM employ
 WHERE sal > 9000;
BEGIN
 for emp_rec in emp_cur
 loop
 dbms_output.put_line (emp_rec.ename || ' ' || emp_rec.sal);
 end loop;
END;
/
```

**Output:**

```
Rani 10000
Ranbir 11000
PL/SQL procedure successfully completed.
```

### 10.5 Parameterized Cursors:

Oracle permits the data that is retrieved from the table be allowed to change according to need. The contents of parameterized cursor will constantly change depending on the value that is passed as parameter. Since the cursor accepts user-defined values into its parameters, thus changing the result set extracted, it is called parameterized cursor.

#### Declaring Parameterized cursor:

Syntax:

```
CURSOR cursor_name IS <SELECT statement>;
```

#### Opening a Parameterized Cursor:

Syntax:

```
OPEN cursor_name(value/variable/expression);
```

Example:

```
/*DISPLAY DETAILS OF STUDENT WITH ROLLNUMBER */
SET SERVEROUTPUT ON
DECLARE
 cursor c(no number) is select * from student
 where rollno = no;
 tmp student%rowtype;
BEGIN
 FOR tmp IN c(14)
```

```
LOOP
dbms_output.put_line('ROLLNO: '||tmp.rollno);
dbms_output.put_line('STUDENT_Name: '||tmp.name);
dbms_output.put_line('TOTAL: '||tmp.total);
dbms_output.put_line('COURSE: '||tmp.course);
END Loop;
END;
/
```

Output:

```
SQL> @PARCUR.SQL
ROLLNO: 14
STUDENT_Name: Gita
TOTAL: 145
COURSE: bca
```

PL/SQL procedure successfully completed.

### Assignment

#### Short Answers (2 marks)

1. Define a cursor in PL/SQL
2. Give the syntax of declaring a cursor
3. List the commands used to work with explicit cursors

#### Long answers(4 or more marks)

1. What is CURSOR? Explain all explicit cursor attributes.
2. Explain all the implicit cursor attributes.
3. Explain the working of implicit cursor with an example
4. Explain the working of explicit cursor with an example
5. Explain the commands necessary to work with cursors?
6. Explain parameterized cursors with an example
7. Explain cursor for loop with an example

CHAPTER 11  
ERROR HANDLING IN PL/SQL

Introduction:

When an SQL statement fails, the Oracle engine is the first to recognize this as an exception condition. The oracle engine immediately tries to handle the exception and resolve it. This is done by raising a built-in exception handler. An exception handler is nothing but a code block in memory that will attempt to resolve the current exception condition. The oracle engine can recognize every exception condition that occurs in the memory. To handle very common and repetitive exception conditions, the oracle engine uses **Named Exception Handlers**.

PL/SQL Errors are pre-defined and are automatically raised by Oracle whenever an error is encountered. Each error is assigned a unique number and a message. In PL/SQL, a warning or error condition is called an exception. Exceptions can be internally defined (by the run-time system) or user defined. Examples of internally defined exceptions include division by zero and out of memory. Some common internal exceptions have predefined names, such as ZERO\_DIVIDE and STORAGE\_ERROR. The other internal exceptions can be given names. When an error occurs, an exception is raised. That is, normal execution stops and control transfers to the exception-handling part of your PL/SQL block or subprogram. Internal exceptions are raised implicitly (automatically) by the run-time system. User-defined exceptions must be raised explicitly by RAISE statements, which can also raise predefined exceptions. To handle raised exceptions, you write separate routines called exception handlers.

Syntax:

EXCEPTION

WHEN<Exception Name> THEN  
<User Defined Action to be carried out>

11.1 Oracle’s Named Exception Handlers

The Oracle engine has a set of predefined Oracle error handlers called Named Exceptions. These error handlers are referenced by their names. Some of the pre-defined Oracle named exception handlers are listed below

|                  |                                                                                                                          |
|------------------|--------------------------------------------------------------------------------------------------------------------------|
| DUP_VAL_ON_INDEX | Raised when an insert or update attempts to create two rows with duplicate values in columns constrained by unique index |
| LOGIN_DENIED     | Raised when an invalid user name/password is used to log onto Oracle                                                     |
| NO_DATA_FOUND    | Raised when a select statement returns zero rows                                                                         |
| NOT_LOGGED_ON    | Raised when PL/SQL issues an Oracle call without being logged onto Oracle                                                |
| PROGRAM_ERROR    | Raised when PL/SQL has an internal problem                                                                               |
| TOO_MANY_ROWS    | Raised when a select statement returns more than one row                                                                 |
| VALUE_ERROR      | Raised when the data type or data size is invalid                                                                        |

|        |                                                      |
|--------|------------------------------------------------------|
| OTHERS | Stands for all other exceptions not explicitly named |
|--------|------------------------------------------------------|

11.2 User-Named Exception Handlers

This technique is used to bind a numbered exception handler to a name using **Pragma Exception\_init()**. This binding of a numbered exception handler, to a name (String) is done in the declare section of the PL/SQL block.

The Pragma action word is a call to a pre-compiler, which immediately binds the numbered exception handler to the name when encountered. The function Exception\_init() takes two parameters – the first is the user defined exception name and the second is the Oracle engine’s exception number. These lines will be included in the declare section of the PL/SQL block.

**Syntax:**  
DECLARE  
<Exception Name> EXCEPTION  
PRAGMA EXCEPTION\_INIT(<Exception Name>,<Error Code>);  
BEGIN  
.....  
EXCEPTION  
WHEN<Exception Name> THEN  
    <Action>  
END;

11.3 User Defined Exception Handlers

11.3.1 User Defined Exception Handlers for Business Rule Validation

In commercial applications, data that is being manipulated needs to be validated against business rules. If the data violates a business rule, the entire record must be rejected. To trap business rules being violated the technique of raising user-defined exceptions and then handling them is used.

User defined error conditions are declared in the DECLARE section of PL/SQL block. In the executable part, a check for the condition is made. If the condition exists, the call to the user defined exception is made using RAISE statement. The exception once raised is then handled in the exception handling section of the PL/SQL code block.

**Syntax:**  
DECLARE  
    <<Exception Name> Exception;  
BEGIN  
SQL statements;  
IF <condition>THEN  
RAISE <Exception Name>;  
END IF;  
EXCEPTION



```
WHEN <Exception Name> THEN {user-defined action}
END;
```

Example:

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
 myex EXCEPTION;
```

```
 i NUMBER;
```

```
BEGIN
```

```
 FOR i IN (SELECT * FROM student)
```

```
 LOOP
```

```
 IF i.rollno = 15 THEN
```

```
 RAISE myex;
```

```
 END IF;
```

```
 END LOOP;
```

```
EXCEPTION
```

```
 WHEN myex THEN
```

```
 dbms_output.put_line('ROLLNO is Filled Already');
```

```
END;
```

```
/
```

Output:

```
SQL> @USEREX.SQL
```

```
ROLLNO is Filled Already
```

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

### Assignment

#### Short Answers (2 marks)

1. What do you mean by an exception handlers?
2. List any two oracle named exceptions.
3. Give the syntax of dealing with exception in PL/SQL.

#### Long answers(4 or more marks)

1. Explain user defined exceptions for business rule validation with an example.
2. Explain user named exception handlers
3. Explain exception handling in PL/SQL

## CHAPTER 12

### STORED PROCEDURES AND FUNCTIONS

#### 12.1 Introduction:

Stored Procedures and functions are PL/SQL database objects that are stored in the Oracle database. They can be invoked or called by any PL/SQL block that appears within an application. Before a procedure or function is stored, the oracle engine parses and compiles the procedure/function. Oracle engine performs the following steps automatically while creating a procedure or function

- Compiles the procedure or function
- Stores the procedure or function in the database

The oracle engine compiles the PL/SQL block. If errors occur during the compilation of procedure or function, an invalid procedure or function gets created. The oracle engine displays the message after creation of that the procedure or function was created with compilation errors. The compilation process does not display any errors. These errors can be viewed by giving the following statement at the SQL prompt

```
SQL> SELECT * FROM USER_ERRORS;
```

#### 12.2 Advantages of using procedure or function

The following are the advantages of using functions or procedures

- a. Security – Stored procedures and functions can help to enforce data security. For example, we can give permission to a stored procedures or function to query a table and grant permission to the user to use the stored procedures or function, rather than the table itself.
- b. Performance – it improves database performance in the following ways:
  - Amount of information sent over network is less
  - No compilation step to execute the code
- c. Memory allocation – the amount of memory used reduces as stored procedures and functions have shared memory capability. Only one copy of stored procedure or function needs to be loaded for execution by multiple users.
- d. Productivity – redundant coding can be avoided thereby increasing the productivity
- e. Integrity - Stored procedures and functions need to be tested only once to guarantee that it returns accurate result.

#### 12.3 Stored Procedures:

A stored procedure or in simple a proc is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section

and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

**General Syntax to create a procedure is:**

```
CREATE OR REPLACE PROCEDURE procedure-name
(<Argument> {IN, OUT, IN OUT}<data type>...)
IS
 Declaration section
BEGIN
 Execution section
EXCEPTION
 Exception section
END;
```

We can pass parameters to procedures in three ways.

1. IN – indicates that the parameter will accept a value from the user
2. OUT – indicates that the parameter will return a value to the user
3. IN OUT - indicates that the parameter will either accept a value from the user or return a value to the user

**Example:**

```
SQL> ed proc1.sql
```

```
/*procedure */
CREATE or REPLACE PROCEDURE pro1(no in number,temp out student%rowtype)
IS
BEGIN
SELECT * INTO temp FROM student WHERE rollno = no;
END;
/
```

```
SQL> @proc1;
Procedure created.
```

```
SQL> ed sqlproc.sql
/*PL-SQL code for implementing procedure*/
SET SERVEROUTPUT ON
DECLARE
 temp student%rowtype;
 no number :=&no;
BEGIN
 pro1(no,temp);
 dbms_output.put_line(temp.rollno||' '||
 temp.name||' '||
 temp.total);
END;
```

```
/
```

**Output:**

```
SQL> @sqlproc.sql;
Enter value for no: 12
old 3: no number :=&no;
new 3: no number :=12;
12 Sriram 235
PL/SQL procedure successfully completed.
```

**12.4 Functions:**

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value. Here, return type in the header section defines the return type of the function. The return data type can be any of the oracle data type like varchar, number etc.

**Syntax for creating a function:**

```
CREATE OR REPLACE FUNCTION <function_name>
(<Argument> {IN }<data type>...)
RETURN return_datatype;
IS
Declaration_section
BEGIN
Execution_section
Return return_variable;
EXCEPTION
exception section
Return return_variable;
END;
```

**Example:**

```
SQL> ED STUDEFUN.SQL
/*create a function*/
CREATE or REPLACE FUNCTION STUDEFUN(no in number) RETURN varchar2
IS
name varchar2(20);
BEGIN
select name into name from student where rollno = no;
return name;
END;
```

```
SQL> @STUDEFUN.SQL;
Function created.
```

```
SQL> ed plfun.sql
/*PL-SQL block to call the function*/
```

```
SET SERVEROUTPUT ON
DECLARE
 no number :=&no;
 name varchar2(20);
BEGIN
 name := studfun(no);
 dbms_output.put_line('Name: ' || name);
END;
```

**Output:**

SQL> @plfun.sql;

Enter value for no: 12

old 2: no number :=&no;

new 2: no number :=12;

Name: Sriram

PL/SQL procedure successfully completed.

**12.5 Procedures versus Functions:**

The differences between a procedure and a function are listed below:

- A function must return a value back to the caller. A function can return only one value to the called PL/SQL block
- By defining multiple OUT parameters in a procedure, multiple values can be passed to the caller. The OUT variable is global in nature. Hence, it can be accessible by any PL/SQL code block.

**Assignment****Short Answers (2 marks)**

1. What do you mean by a stored procedure?
2. What do you mean by a function in PL/SQL?
3. Differentiate between function and procedure.
4. Give the syntax of creating a function in PL/SQL.
5. Give the syntax of creating a procedure in PL/SQL.
6. What are the ways in which you can pass parameters to procedures in PL/SQL?

**Long Answers (4 marks and above)**

1. Explain creating a function in PL/SQL with an example
2. Explain creating a procedure in PL/SQL with an example
3. What are the advantages of using functions and procedures in PL/SQL

## CHAPTER 13

### ORACLE PACKAGES

#### Introduction:

A package is an Oracle object, which holds other objects within it. PL/SQL packages are schema objects that groups logically related procedures, functions, constants, variables and cursors. It is a way of creating generic, encapsulated and reusable code. A package once written and debugged is compiled and stored in Oracle's system tables held in Oracle database. Packages contain PL/SQL blocks of code, which have been written to perform some process entirely on their own. These PL/SQL blocks do not require any input from other PL/SQL blocks of code.

#### 13.1 Components of an Oracle Package:

A package has usually two components – **specification** and a **body**.

##### 13.1.1 Package Specification:

The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms. All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

Package specification contains

- Name of the package
- Names of the data types of any arguments
- This declaration is local to the database and global to the package

The syntax for package specification is as follows:

```
CREATE PACKAGE <package-specification-name> IS
<function declaration>
<procedure declaration >
END<package-specification-name>;
```

##### 13.1.2 Package Body:

The body of a package contains the definition of the public objects that are declared in the specification. The package body has the codes for various methods declared in the package. The CREATE PACKAGE BODY Statement is used for creating the package body. The syntax for creating package body is as follows:

```
CREATE PACKAGE BODY <package-name> AS
<function definition >
<procedure definition>
```

```
END<package-name>;
```

### 13.2 Advantages of Packages

Packages offer the following advantages

- Packages enable organizing applications as efficient modules
- It allows granting of privileges efficiently
- They enable overloading of procedures and functions, if required
- They promote reusability of code
- They improve performance by loading multiple objects into the memory at once.

### 13.3 Creating a Package:

The first step in creating a package is to create its specification. The specification declares the objects that are contained in the body of the package. A package can include functions and procedures. After the specification is created, the body of the package needs to be created. The body of the package is a collection of detailed definitions of the objects that were declared in the specification.

Example:

Display the student name, course and total by implementing package

```
/*Creating package specification*/
```

```
SQL> ed sampkg
```

```
CREATE or REPLACE PACKAGE pkg1
```

```
AS
```

```
PROCEDURE janpro1
```

```
 (no in number);
```

```
FUNCTION janfun1
```

```
 (no in number)
```

```
 RETURN number;
```

```
END pkg1;
```

```
/
```

```
SQL> @sampkg;
```

Package created.

```
/*Creating body of package*/
```

```
SQL> ed pkgbody
```

```
CREATE or REPLACE PACKAGE BODY pkg1
```

```
AS
```

```
 FUNCTION janfun1(no in number) return number
```

```
 IS
```

```
 tot number;
```

```
 BEGIN
```

```
 select total into tot from student where rollno = no;
```

```
 return tot;
```

```
 END;
```

```
 PROCEDURE janpro1(no in number)
```

```
IS
temp1 student.name%type;
temp2 student.course%type;
BEGIN
 select name,course into temp1, temp2 from student where rollno = no;
 dbms_output.put_line('Student Name:' || temp1);
 dbms_output.put_line('Course Name:' || temp2);
END;

END pkg1;
/
SQL> @pkgbody;
Package body created.

/*PL/SQL code for implementing package*/
SQL> ed plpkg
SET SERVEROUTPUT ON
DECLARE
 no number := &no;
 m number;
BEGIN
 pkg1.janpro1(no);
 m := pkg1.janfun1(no);
 dbms_output.put_line('Total: ' || m);
END;
/
SQL> @plpkg;
Enter value for no: 12
old 2: no number := &no;
new 2: no number := 12;
Student Name:Sriram
Course Name:bca
Total: 235
```

**Assignment:****Short Answers(2 marks)**

1. Define a package in PL/SQL
2. What are the components available in the package specification of PL/SQL
3. Give the syntax of a package specification
4. Give the syntax of creating a package body

**Long Answers(4 marks and above)**

1. What are the advantages of packages
2. How do you work with packages in PL/SQL? Explain with an example
3. Explain with an example the working of package specification



CHAPTER 14  
DATABASE TRIGGERS

14.1 Introduction

Database triggers are database objects created on the client side and stored on the server in the Oracle’s engine system table. They consists of the following sections

- A named database event
- A PL/SQL block that will execute when the event occurs

The occurrence of the database event is strongly bound to table data being changed. The Oracle engine allows the definition of procedures that are implicitly executed when an insert, update or delete operation is issued at the SQL prompt or through an application. These procedures are called **database triggers**. Triggers are standalone because they are fired implicitly by the Oracle engine.

14.2 Use of Database Triggers

The advantages of triggers are

- A trigger can permit DML statements against the table only if they are issued at a particular period of time
- A trigger keeps an audit trail of a table (i.e. stored records/modified/deleted records) along with the operation performed and the time on which the operation was performed
- It can be used to prevent invalid transactions
- Enforce complex security authorizations

14.3 Database Triggers versus Procedures

| Database Trigger                                                                    | Procedure                                                   |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------|
| Triggers do not accept parameters                                                   | Procedures accept parameters                                |
| A trigger is implicitly by the oracle engine itself upon modification of table data | To execute a procedure, it is explicitly called by the user |

14.4 Applying Database Triggers

A trigger consists of three parts. They are

1. A triggering event or statement
2. A trigger restriction
3. A trigger action

Triggering Event or Statement:

It is an SQL statement that causes a trigger to be fired. It can be INSERT, UPDATE or DELETE statement for a specific table

Trigger Restriction

A trigger restriction specifies a Boolean expression that must be **TRUE** for a trigger to fire. It is an option available for triggers that are fired for each row. Its function is to conditionally control the execution of a trigger. A trigger restriction is specified using the **WHEN** clause

### Trigger Action

A trigger action is the PL/SQL code to be executed when a triggering statement is encountered and any trigger restriction evaluates to TRUE. For row triggers, the statements in the PL/SQL block have access to the column values of the current row being processed

### 14.5 Types of Triggers

There are two types of triggers based on the level at which it is fired. **Row trigger** – A row trigger is fired each time a row in a table is affected by the triggering statement. For example, if an update statement modifies multiple rows in a table, a row trigger is fired once for each row affected by the update statement. If no rows are affected, the trigger is not executed at all. Row triggers should be used when some processing is required whenever a triggering statement affects a single row in a table.

**Statement trigger** – A statement trigger is fired once on behalf of the triggering statement, independent of the number of rows being affected by the trigger. Statement triggers are used when a triggering statement affects the rows in the table but the processing is independent of the number of rows affected.

### 14.6 Creating a Trigger

#### Syntax for creating a Trigger:

```
CREATE OR REPLACE TRIGGER <trigger_name>
{BEFORE , AFTER }
{INSERT, UPDATE, DELETE [OF col_name]}
ON <table_name>
[REFERENCING OLD AS old NEW AS new]
[FOR EACH ROW WHEN <condition>]
DECLARE
 Variable declaration;
BEGIN
 PL/SQL statements;
END;
```

#### Example:

/\*Create a trigger to insert the names of employees in CAPITALS\*/

```
SQL> ed samtrg
CREATE or REPLACE TRIGGER trg1
BEFORE
INSERT ON employ
for each row
BEGIN
```

```
:new.ename := upper(:new.ename);
END;
/
```

```
SQL> @samtrg;
```

Trigger created.

```
SQL> insert into employ
2 values('&empno','&ename',&sal,'&deptno');
Enter value for empno: E008
Enter value for ename: ranjit
Enter value for sal: 7000
Enter value for deptno: D004
old 2: values('&empno','&ename',&sal,'&deptno')
new 2: values('E008','ranjit',7000,'D004')
```

1 row created.

**Output:**

```
SQL> select ename from employ;
ENAME
```

-----

```
Gita
Raja
Rani
Jaya
Ranbir
Rajni
Sita
RANJIT
```

8 rows selected.

**14.7 Deleting a Trigger**

A trigger can be deleted using the DROP command.

Syntax:

```
DROP TRIGGER <trigger_name>;
```

**Example:**

```
SQL> drop trigger trg2;
Trigger dropped.
```

**14.8 RAISE\_APPLICATION\_ERROR PROCEDURE**

**RAISE\_APPLICATION\_ERROR** is a built-in procedure in oracle which is used to display the user-defined error messages along with the error number whose range is in between -20000 and -20999. Whenever a message is displayed using **RAISE\_APPLICATION\_ERROR**, all previous transactions which are not committed within the PL/SQL Block are rolled back automatically. **RAISE\_APPLICATION\_ERROR** raises an exception but does not handle it. **RAISE\_APPLICATION\_ERROR** is used for the following reasons:

- a. To create a unique id for a user-defined exception
- b. To make user defined exception look like an Oracle error

The General Syntax is:

**RAISE\_APPLICATION\_ERROR (error\_number, error\_message);**

- The Error number must be between -20000 and -20999
- The Error\_message is the message you want to display when the error occurs.

Steps to be followed for using **RAISE\_APPLICATION\_ERROR** are

1. Declare a user-defined exception in the declaration section.
2. Raise the user-defined exception based on a specific business rule in the execution section.
3. Finally, catch the exception and link the exception to a user-defined error number in **RAISE\_APPLICATION\_ERROR**.

**Example:**

```
CREATE or REPLACE TRIGGER trg2
AFTER
DELETE ON employ
for each row
```

```
BEGIN
IF :old.empno = 'E001' THEN
 raise_application_error(-20015, 'Do Not Delete this Row');
END IF;
END;
/
```

Output:

```
SQL>delete from emp1 where eno = 1;
Error Code: 20015
Error Name: Do Not Delete this Row
```

### 14.9 Generating Primary key using Database Triggers

In a multi user environment, when stat entry operators create and enter primary key to uniquely identify records, it will always result in a large number of records being rejected due to duplicate values being keyed due to human error. In order to avoid this situation, a primary key can be generated automatically by not allowing the user to type the value for primary key. In this case primary key is system generated. The various approaches to generate primary key are

- Use a look up table that stores the last primary key value
- Use the MAX function

- Use a Sequence

### **Assignment**

#### **Short Questions (2 marks)**

1. Define a trigger in PL/SQL
2. Give the syntax of creating a trigger.
3. What is the purpose of raise\_application\_error?
4. List the various methods of generating primary key using database triggers.
5. List the various types of triggers
6. Differentiate between database trigger and procedures
7. List the various components of triggers.

#### **Long Questions (5 & above marks)**

1. Explain the general syntax of creating a trigger with an example.
2. What is trigger? What are the different parts of triggers? Explain.
3. Explain the purpose of raise\_application\_error with an example.

## CHAPTER 15

### INDEX STRUCTURES FOR FILES

#### 15.1 Introduction

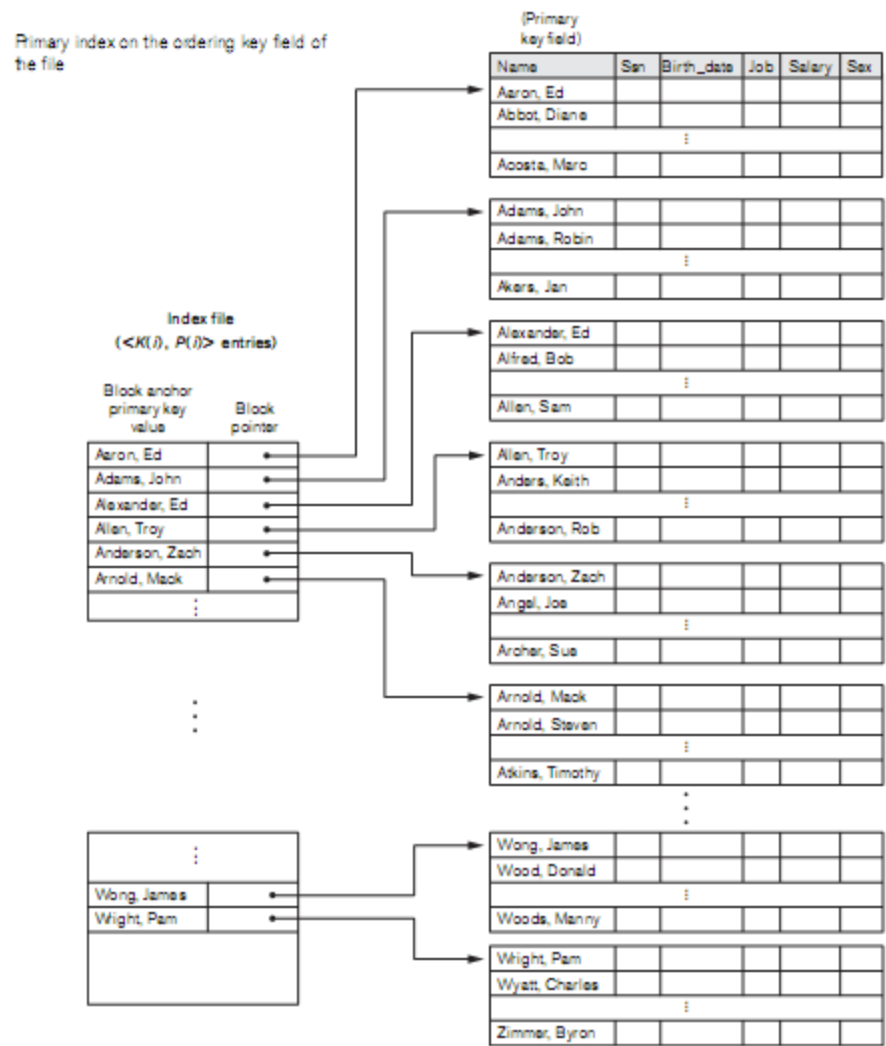
Indexes are auxiliary access structures that are used to speed up the retrieval of records in response to certain search conditions. The index structures are additional files on disk that provide secondary access paths, which provide alternative ways to access the records without affecting the physical placement of records in the primary data file on disk. They enable efficient access to records based on the indexing fields that are used to construct the index. Basically, any field of the file can be used to create an index and multiple indexes on different fields – as well as indexes on multiple fields—can be constructed on the same file. To find a record or records in the data file based on a search condition on an indexing field, the index is searched, which leads to pointers to one or more disk blocks in the data file where the required records are located. The different types of single-level ordered indexes – primary, secondary, and clustering index.

A primary index is specified on the ordering key field of an ordered file of records. Ordering key field is used to physically order the file records on disk, and every record has a unique value for that field. If the ordering field is not a key field—that is, if numerous records in the file can have the same value for the ordering field – another type of index, called a clustering index, can be used. The data file is called a clustered file. In the latter case, a file can have at most one physical ordering field, so it can have at most one primary index or one clustering index, but not both. A third type of index, called a secondary index, can be specified on any non-ordering field of a file. A data file can have several secondary indexes in addition to its primary access method.

#### 15.2 Primary Index

A primary index on an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field – called the primary key of the data file and the second field is the pointer to the disk block. There is one index entry in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to that block as its two fields.

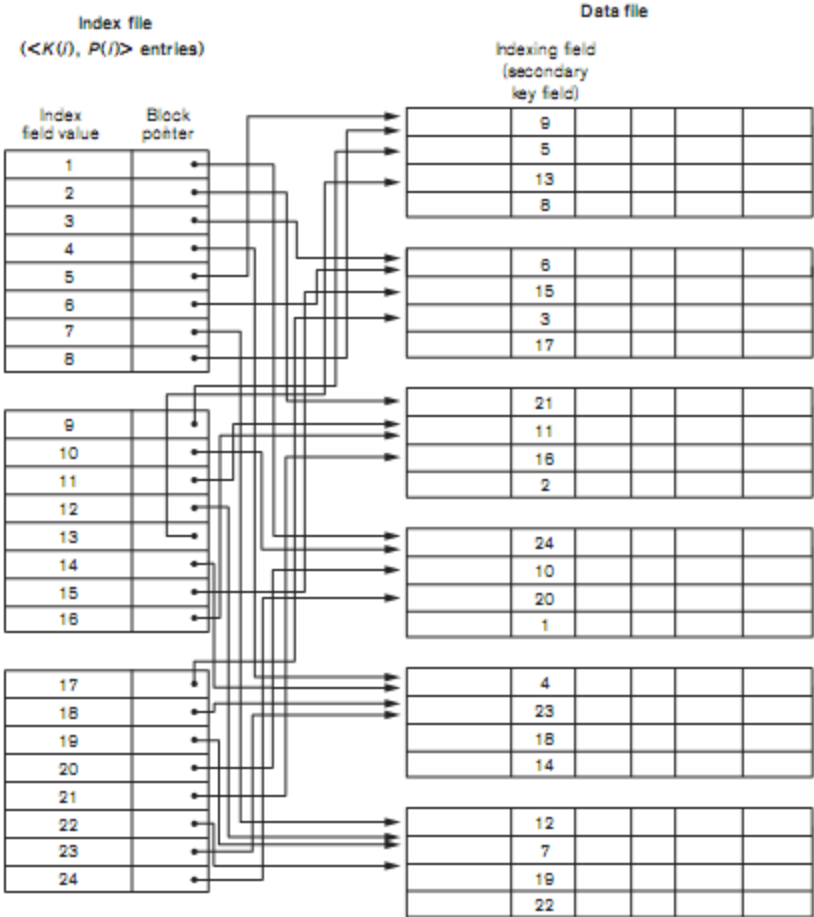
The total number of entries in the index is equal to the number of disk blocks in the ordered data file. The first record in each block of the data file is called the **anchor record** of the block or **block anchor**. Indexes can be characterized as either dense or sparse. A **dense index** has an index entry for every search key value i.e. every record in the data file. A **sparse** or **non-dense index** has index entries for only some of the search values. A primary index is a sparse index since it includes an entry for each disk block of the data file and the keys of the anchor record.



15.3 Secondary Index

A secondary index provides a secondary means of accessing a file for which some primary access already exists. The secondary index may be on a field which is a candidate key and has a unique value in every record or a non-key with duplicate values. The index is an ordered file with two fields. The first field is of the same data type as the non ordering field of the data file called the indexing field. The second is either a block pointer or a record pointer. There can be many secondary indexes on the same file. When the secondary index access structure on a key field has distinct value for every record is also referred to as **secondary key**. Here there is one index for each record in the data file, which contains a value of the secondary key for the record and a pointer either to the record or the block in which the record is stored. Such index is said to be **dense**. A secondary index needs more storage space and a longer search time than primary index because of large number of entries.

Adense secondary index (with block pointers) on a nonordering key field of a file.

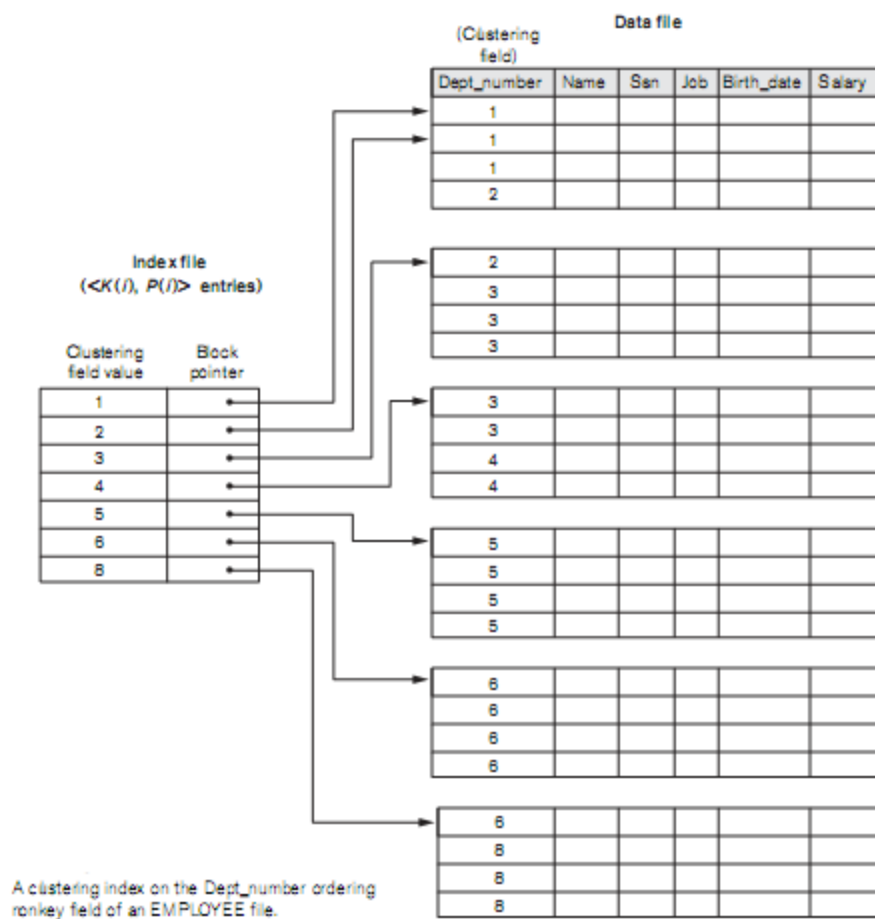


15.4 Clustering Index

If the records of a file are physically ordered on a non-key field – which do not have a distinct value for each record – that record is called the clustering field. We can create an index called clustering index, to speed up the retrieval of records that have the same value for clustering field. This differs from primary index, which requires that the ordering field of the data field of the data file must have a distinct value for each record.

A clustering index is also an ordered file with two fields: the first is of the same type as the clustering field of the data file and the second field is a block pointer. There is one entry in the clustering index for each distinct value of the clustering field, containing the value and a pointer to the first block in the data file that has a record as a with that value as its clustering field. A clustering index is another example for a **nondense index**; because it has an entry for every distinct value of the indexing filed which is a non key by definition and hence duplicate values rather than for every record in the file.

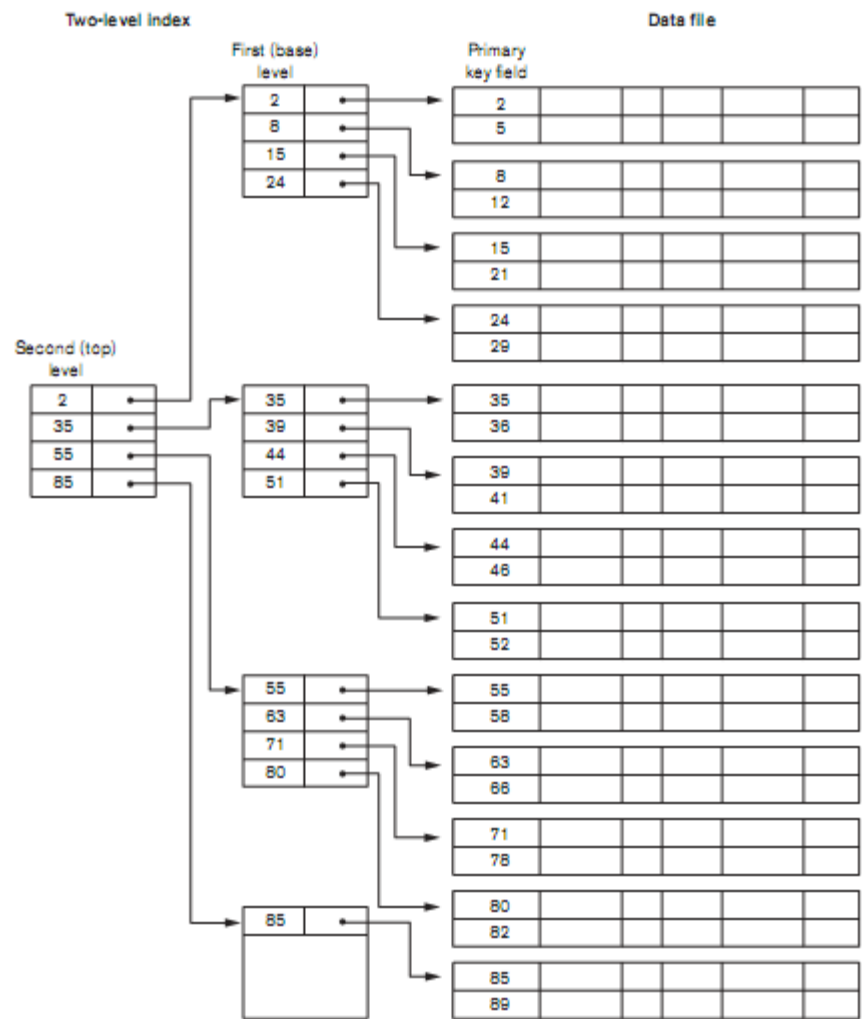




## 15.5 Multilevel Index

A multilevel index considers an index file which is referred to as the base level or first level index as an ordered file with a distinct value for each  $K(i)$ . Hence, we create a primary index for the first level; this index to the first level is called the second level of the multilevel index. Since the second level is a primary index, we can use block anchors so that the second level has one entry for each block of the first level. In this way  $n$  levels of indexes can be created for one data file and hence the name.

A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.



**Credit Based second semester B.C.A. Degree Examination, April/May 2015**  
**(New Syllabus - 2012-2013 Batch Onwards)**  
**DATABASE CONCEPTS AND ORACLE**

***Note:** Answer **any ten** questions from Part A and **anyone full** question from each Unit of Part B.*

**PART A**

1. a) Define Metadata and System catalog.  
b) What is database?  
c) What is a derived attribute? Give example  
d) What is data stripping?  
e) What do you mean by degree of a relation?  
f) What is NULL value in an attribute?  
g) State with example how do you delete a record in SQL.  
h) What is DDL?  
i) What is dual table in SQL?  
j) List any two Oracle named exceptions  
k) How do you assign values to variables in SQL?  
l) Name the two forms of comments in PL/SQL.

**PART B**

**UNIT-I**

2. a) What are the responsibilities of database administrator  
b) Write any five advantages of DBMS.  
c) Explain various symbols used in ER diagrams (5+5+5)
3. a) Describe the Three schema architecture of DBMS with diagram.  
b) What is data independence? Explain.  
c) Explain any two database interfaces (5+5+5)

**UNIT-II**

4. a) Explain any five operations on files.  
b) State the usage of update operations on relational model  
c) Explain the types of hashing techniques (5+5+5)
5. a) Explain the complete set of relational algebra operations  
b) Explain SELECT and PROJECT operations with examples  
c) Write a note on magnetic disks (5+5+5)

**UNIT-III**

6. a) List and explain five data types in Oracle.
  - b) Explain update command in Oracle with an example
  - c) Explain ORDER BY and GROUP BY clause in SQL (5+5+5)
- 
7. a) Write SQL queries for the following operation on EMP table.  
EMP (ENO,NAME,DEPT,BASIC,DATE\_JOIN)
  - i) Retrieve the details of employees of Accounts department.
  - ii) Add a column DESIG of data type varchar and size 15.
  - iii) Increase the basic salary of employees by 5% belonging to "Admin" department.
  - iv) List the employees details having b as second character in their names.
  - b) How do you insert data from one table to another? Explain with an example
  - c) Explain pattern matching predicate with an example (5+5+5)

**UNIT-IV**

8. a) Explain the PL/SQL block structure with an example.
  - b) Explain the conditional statements in PL/SQL with examples.
  - c) What is a Trigger? Explain the types of Trigger. (5+5+5)
- 
9. a) With syntax and example explain any two looping statements in PL/SQL.
  - b) What is a function? How do you create a function? Explain.
  - c) What are the advantages of PL/SQL. (5+5+5)

