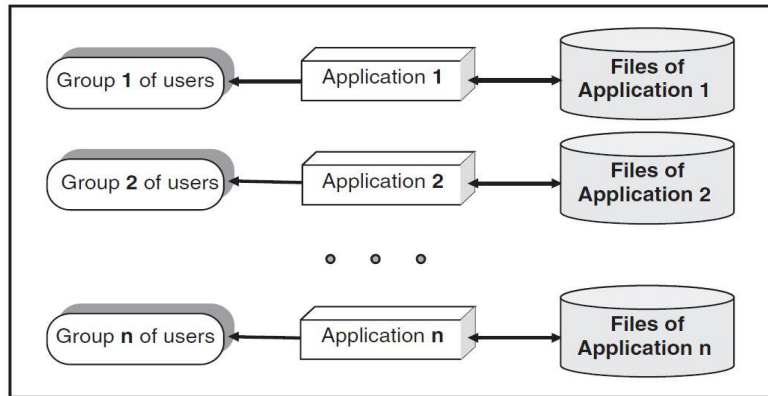


# UNIT - I

Q) Explain about file processing system? What is conventional file processing system? What are the drawbacks of CFP'S?

## File-Based System

Prior to DBMS, file system provided by OS was used to store information. In a file-based system, we have collection of application programs that perform services for the end users. Each program defines and manages its own data.



## **Drawbacks of File-Based System/Drawbacks of File Processing Systems**

1. **Uncontrolled redundancy of data:** In CFP'S each subsystem is having a separate set of files without data sharing. It creates lot of data redundancy and wastage of memory, time and money.
2. **Program-Data Dependence:** File descriptions are stored within each application program that accesses a given file. If any change in the file description (like file type, size, add a field, delete a field etc) is found then rewrite the program to accommodate these changes.
3. **Duplication of Data:** Each subsystem requires separate application program to process the data. These applications programs are developed independently so that the data is duplicated between applications. It is also possible that the same data item may have different names in different files, or the same name may be used for different data items in different files. If any change in the data these changes are done manually in all the files. This results loss of data integrity.
4. **Limited data Sharing:** Each application has its own private files with little opportunity to share data outside their own applications. A requested report may require data from several incompatible files in separate systems.
5. **Low Programming Productivity:** Each new application requires the developer to start from scratch by designing new file formats and descriptions.
6. **Excessive Program Maintenance:** In CFP'S for any modification in the data then the program must be modified. This increases the program maintenance.
7. **Integrity Problem:** The problem of integrity is the problem of ensuring that the data in the database is accurate.

## **Some of the top File Processing Systems are:**

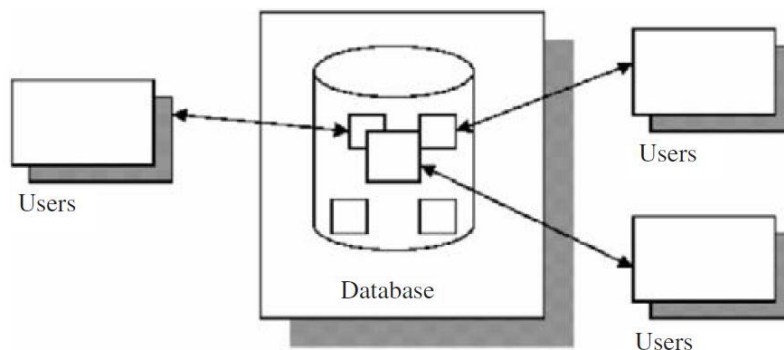
RoboRealm 1.0 , Numerology 369 1.0, Unix2Dos 1.1, Data Master 2000 10.6.0.107, Eym Barcode Reader OCX 2.4 , Captcha

## Data and Information

Data are raw facts that constitute building block of information. Data are the heart of the DBMS. It is to be noted that all the data will not convey useful information. Useful information is obtained from processed data.

## Database

A database is a well-organized collection of data that are related in a meaningful way, which can be accessed in different logical orders. The database should contain all the data needed by the organization as a result, a huge volume of data, the need for long-term storage of the data, and access of the data by a large number of users generally characterize database systems.



## Database Management System

A database management system (DBMS) consists of collection of interrelated data and a set of programs to access that data. It is a software that is helpful in maintaining and utilizing a database.

### **DBMS consists of:**

- A collection of interrelated and persistent data. This part of DBMS is referred to as database (DB).
- A set of application programs used to access, update, and manage data. This part constitutes data management system (MS).
- A DBMS is general-purpose software i.e., not application specific. The same DBMS (e.g., Oracle, Sybase, etc.) can be used in railway reservation system, library management, university, etc.
- A DBMS takes care of storing and accessing data, leaving only application specific tasks to application programs.

## **Q. Explain the objectives of DBMS?**

The main objectives of database management system are data availability, data integrity, data security, and data independence

### **1. Data Availability**

Data availability refers to the fact that the data are made available to wide variety of users in a meaningful format at reasonable cost so that the users can easily access the data.

### **2. Data Integrity**

Data integrity refers to the correctness of the data in the database

### **3. Data Security**

Data security refers to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular data at the same time, the DBMS must not allow them to make conflicting changes.

### **4. Data Independence**

DBMS allows the user to store, update, and retrieve data in an efficient manner. DBMS provides an “abstract view” of how the data is stored in the database.

## Evolution of Database Management Systems

Apollo moon-landing process was started in the year 1960. At that time, there was no system available to handle and manage large amount of information. As a result, North American Aviation which is now popularly known as Rockwell International developed software known as Generalized Update Access Method (GUAM). In the mid-1960s, IBM joined North American Aviation to develop GUAM into Information Management System (IMS). IMS was based on Hierarchical data model.

In the mid-1960s, General Electric released Integrated Data Store (IDS). IDS were based on network data model. Charles Bachmann was mainly responsible for the development of IDS.

Conference on Data System Languages formed Data Base Task Group (DBTG) in 1967. DBTG specified three distinct languages for standardization. They are Data Definition Language (DDL and DML)

The network and hierarchical data models developed during that time had the drawbacks of minimal data independence, minimal theoretical foundation, and complex data access

In 1970, Codd of IBM published a paper titled “A Relational Model of Data for Large Shared Data Banks” in Communications of the ACM, vol. 13, No. 6, pp. 377–387, June 1970.

Codd’s paper, System R project was developed during the late 1970 by IBM San Jose Research Laboratory in California The outcome of System R project was the development of Structured Query Language (SQL)

In 1980s IBM released two commercial relational database management systems known as DB2 and SQL/DS and Oracle Corporation released Oracle

In recent years, two approaches to DBMS are more popular, which are Object-Oriented DBMS (OODBMS) and Object Relational DBMS

The chronological order of the development of DBMS is as follows:

- Flat files – 1960s–1980s
- Hierarchical – 1970s–1990s
- Network – 1970s–1990s
- Relational – 1980s–present
- Object-oriented – 1990s–present
- Object-relational – 1990s–present
- Data warehousing – 1980s–present
- Web-enabled – 1990s–present

Early 1960s. Charles Bachman at GE created the first general purpose DBMS Integrated Data Store. It created the basis for the network model which was standardized by CODASYL (Conference on Data System Language).

Late 1960s. IBM developed the Information Management System (IMS). IMS used an alternate model, called the Hierarchical Data Model.

1970. Edgar Codd, from IBM created the Relational Data Model. In 1981 Codd received the Turing Award for his contributions to database theory. Codd Passed away in April 2003.

1976. Peter Chen presented Entity-Relationship model, which is widely used in database design.

1980. SQL developed by IBM, became the standard query language for databases. SQL was standardized by ISO.

1980s and 1990s. IBM, Oracle, Informix and others developed powerful DBMS.

## **Q.What are the different types of Database? Or Explain the Classification of Database Management System?**

Database is a collection of inter-related data items that can be processed by one or more application systems. It can be classified according to the number of users, the database locations and data usage.

### **According to Number of users:**

A database can be operated by more number of users. Depending upon number of user use the database it can be classified as single user or multi-user database management system. A single user database supports only one user at a time. For example A,B,C are the users of a database then if user A is using the database user B and C must wait until user A complete the work/ a multi-user database supports multiple users at the same time. If multi-user database supports a small number of users or a specific department within an organization, it is called a workgroup database. If the database is used by the entire organization and supports many users across many departments, the database is known as an enterprise database.

### **According to database location:**

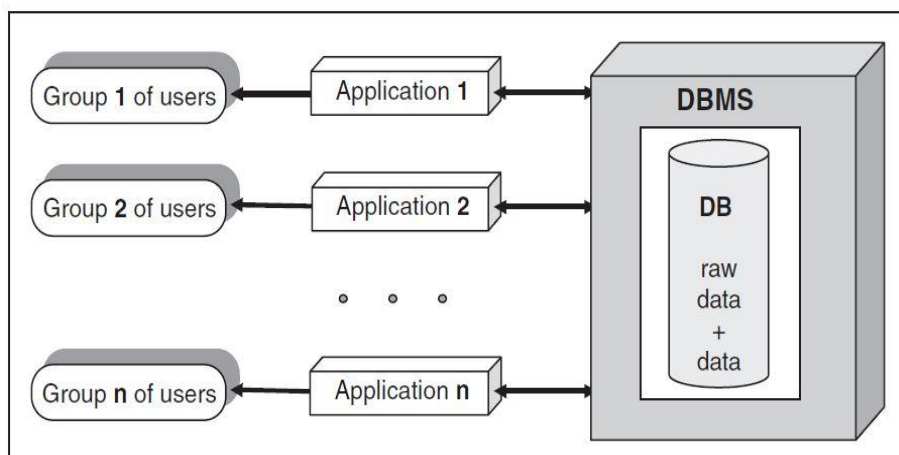
Databases are classified according to the location of the database are centralized database and distributed database. A database that supports data located at a single site is called a centralized database. A database that supports data distributed across several different sites is called a distributed database.

### **According to data usage:**

Databases are classified according to data usage as Operational databases and data warehouse. A database that is designed primarily to supports a company's day-to-day operations is classified as an operational database (transactional or production database). Data warehouse used to generate information required to make tactical or strategic decisions depending on stored data. Such decisions typically require extensive "data massaging"(data manipulations) to extract information to formulate pricing decisions, Sales forecasts, market positioning and so on.

## **Q) Define DBMS? What are the advantages of DBMS?**

DBMS is defined as "collection of programs used to maintain the structure of the database and control the access to the data stored in the database". It is a collection of database, database utilities, data dictionary, database developers and database administrator.



**Data access through DBMS**

## **Advantages of DBMS**

DBMS is defined as “collection of program used to maintain the structure of the database and control the access to the data stored in the database”. It is a collection of database, database utilities, data dictionary, database developers and database administrator.

1. Reduced data redundancy
2. Consistency of data
3. Flexibility of file system
4. Enhanced data sharing
5. Better enforcement of standards
6. Reduced program maintenance
7. High programmer productivity

### **1.Reduced data redundancy:**

In DBMS data is stored in a single place so that the duplication of data is reduced. If duplication is reduced the data redundancy is reduced.

### **2.Consistency of Data:**

If the data redundancy is reduced then the consistency( the presence of uniform data across the whole database) of data is increased. If the consistency is increased then data integrity is increased.

### **3.Flexibility of file system:**

In DBMS data is designed in bottomup approach. In this the end user requirements are studied to develop a database. In future development any changes required in the report are done very easily.

### **4.Enhanced data sharing:**

It is the corner stone of the entire database approach. Same file can be shared between different users and applications which reduces redundancy and inconsistency of the data. This is possible because of the centralized approach of the database.

### **5.Better enhancement ofstandard:**

Generally inconsistencies in the database occur due to improper enforcement of standards in the database design. In DBMS the database is designed at a time to maintain organizational standards like defining field names, with and type etc.

### **6.Reduced program maintenance:**

In DBMS there is less dependency between data and programs. So there is no need to change the program if data is changed. These programs are developed under the coordination of the DBA to follow integrated approach in program writing.

### **7.Increase programmer productivity:**

It is a measure of time taken to develop an application. The productivity of the programmer varies with the time because of the flexibility of the data, better enforcement of standards, low program maintenance. The programmer productivity will drastically increase in DBMS

## Q. Explain the levels of data abstraction? Or What is 3-Schema architecture? Or Explain ANSI/SPARC Data Model?

### Levels of Data Abstraction (or) Ansi/Sparc Data Model

Data abstraction is a process of representing the essential features without including implementation details. In the early 1970 the American National Standards Institute(ANSI) Standards Planning and Requirements Committee(SPARC) defines a framework of modeling based on the degree of data abstraction. They are

1. Physical level or internal level
2. Logical level or conceptual level
3. View level or external level

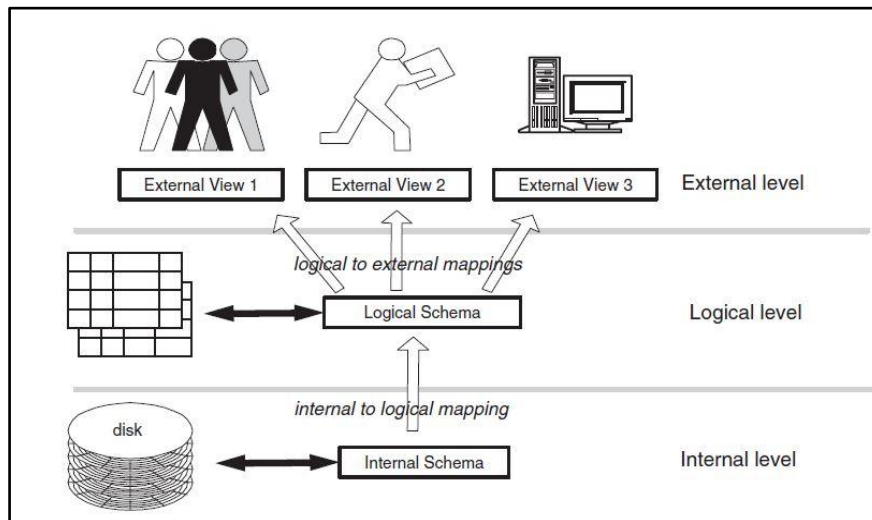
**1. Physical Level:**The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low level data structures.

- Storage allocation e.g. B-trees, hashing etc.
- Access paths e.g. specification of primary and secondary keys, indexes and pointers and sequencing.
- Data compression and encryption techniques.

**2. Logical Level:**This level of abstraction describes what data are stored in the database and what relationships exist among those data. Thus it also describe structures but separating them in small parts.

**3. View Level:** This level represents only a part of the entire database. Even though logical level is simpler than physical level, view level is required because a database may have tons of data all of which are not useful to all users.

- **Data instances:** The collection of information stored in the database at a particular moment is called an instance of the database.
- **Database Schema:** The overall design of the database is called the database schema.

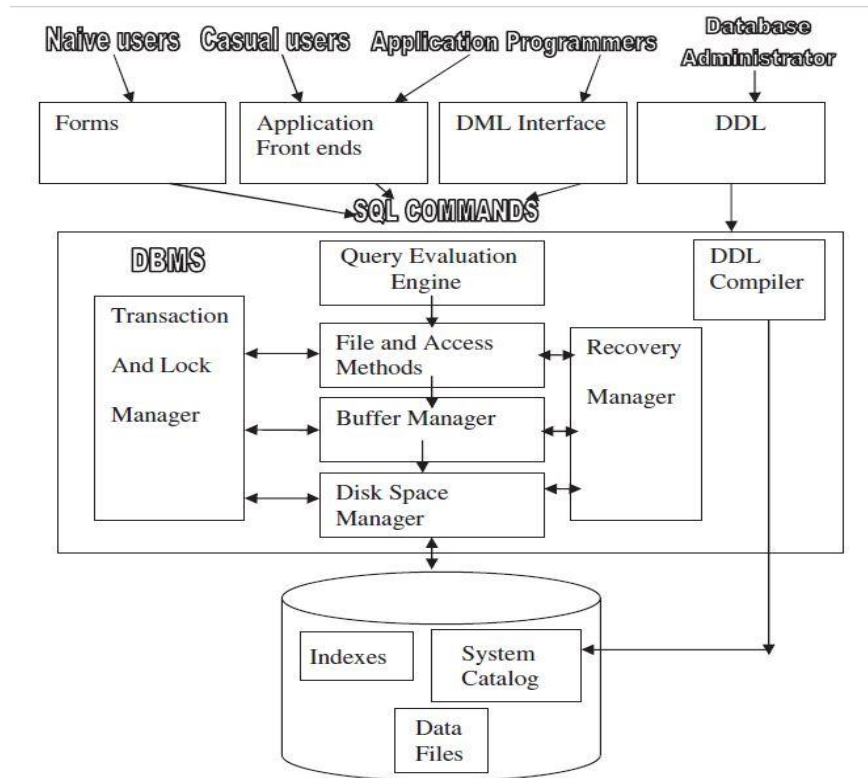


### Q) Define DBMS? What are the components of DBMS environment?

DBMS can be defined as “collection of programs which are used to maintain the structures of the database and control the access to the data stored in the Database”

The components of DBMS environment are

1. Hardware
2. Software
3. People
4. Procedures
5. Data



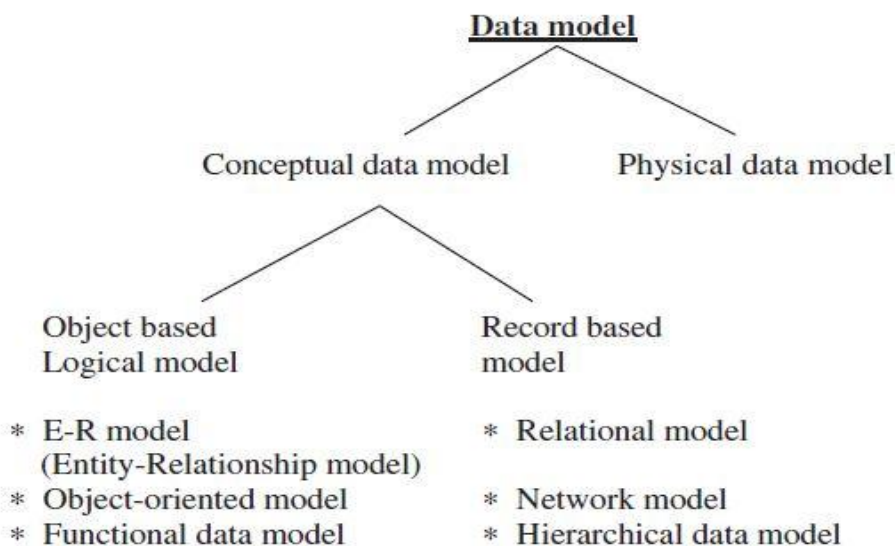
1. **Hardware:** It represents all physical devices like computers, storage devices, printers etc.
2. **Software:** Three types of software's are needed they are
  - a. **Operating System:** It is used to manage all hardware components. For example windows NT, UNIX, Linux etc.
  - b. **DBMS software:** It is used to manage the database. For Example Oracle, SQL server etc.
  - c. **Application programs:** It is used to access and manipulate data in the DBMS. It is used to generate reports. Utilities software tools used to help manage the database system computer components. For example GUI tools used to design database structure, control database access and monitor database.
3. **People:** There are five types of database users they are
  - a. **System Administrator:** See the database system general operations
  - b. **Database Administrator:** DBA manage the DBMS and ensure that the database is functioning properly.
  - c. **Database Designers:** Design the database structure
  - d. **System analysts and Programmers:** Design and implement the application programs.
  - e. **End users:** These are the peoples who use the application programs to run the organization's daily operations.
4. **Procedures:** These are the instructions and business rules used to design and use the database system.
5. **Data:** It represents actual data to be stored in the database for a company
  - a. The data in the database is well organized (structured)
  - b. The data in the database is related
  - c. The data are accessible in different orders without great difficulty



## Q. Explain about different types of Data Models?

### Data Models

Data model is a collection of conceptual tools for describing data, relationship between data, and consistency constraints. Data model describe the structure of the database. A data model provides a way to describe the design of a database at the physical, logical, and view levels.



### Entity-Relationship Model

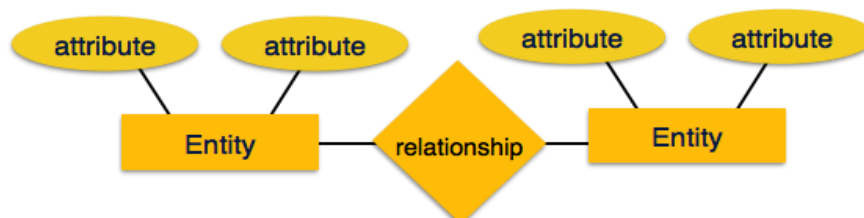
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on –

- **Entities** and their attributes.
- **Relationships** among entities.

These concepts are explained below.



- **Entity** – An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a College database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** – The logical association among entities is called **relationship**. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

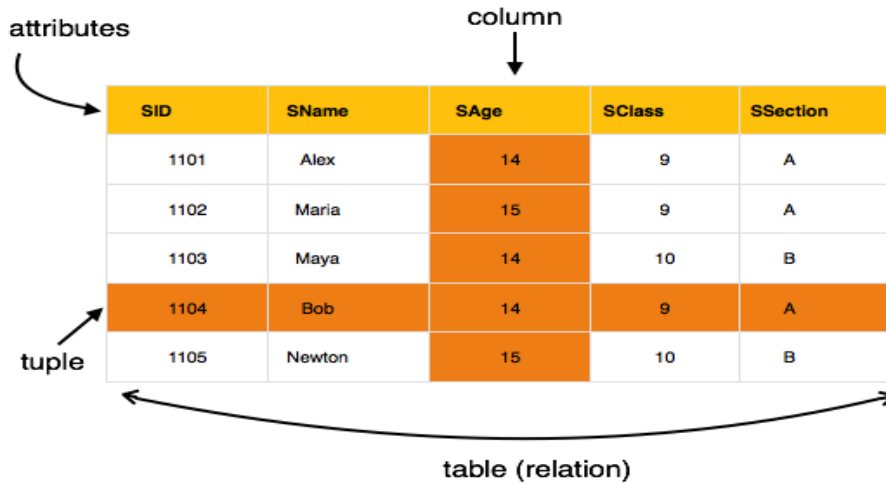
Mapping cardinalities –



- one to one
- one to many
- many to one
- many to many

### Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.



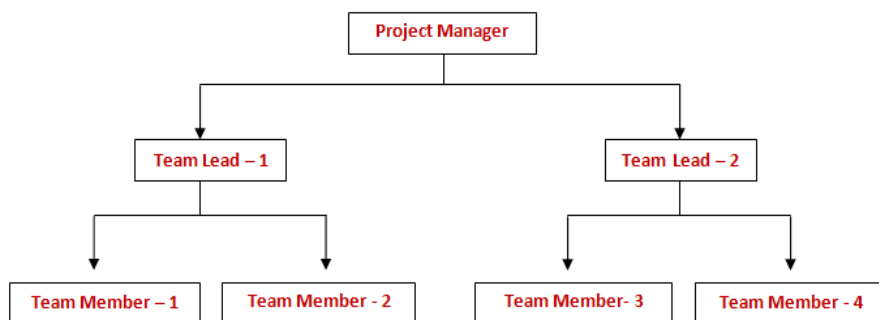
The main highlights of this model are –

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

### Hierarchical Data Model

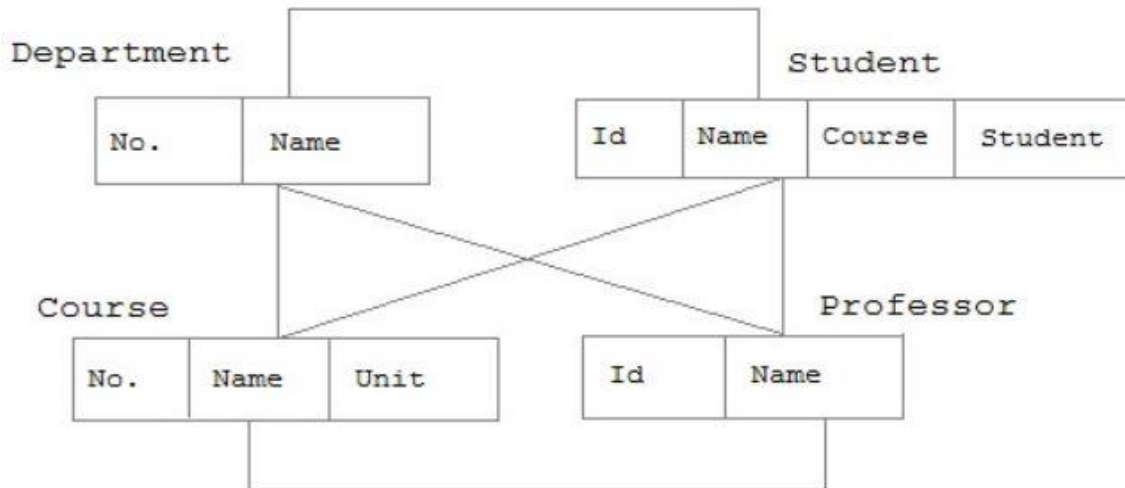
Hierarchical data model was developed in the year 1960 to manage large amount of data for manufacturing projects like Apollo rocket landed on the moon in 1969. It contains set of nested relationships having one to many or one to one association. It does not support many to many associations.

The structure of this model is represented in the form of a tree data structure. The root segment is the parent for the entire branches comedown from the root segment.



## Network Data Model

The network model invented by Charles Bachman and developed in the year 1969. The structure of this model is represented in the form of a graph. It allows multiple records to be linked to the same owner file. In addition, the relationship that the information has in the network database model is defined as many-to-many relationship because one owner file can be linked to many member files and vice versa



## Object-Oriented model

In this model both data and their relationships are in a single structure called object. Each entity represents relationship between facts within the object. It also contain operations performed on it like change data value, find specific data value, print data value etc

Components of OO Model:

1. An object is an abstraction of a real-world entity
2. Attributes describe the properties of an object
3. Object that share similar characteristics are grouped in classes
4. Classes are organized in a class hierarchy
5. Inheritance is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it.

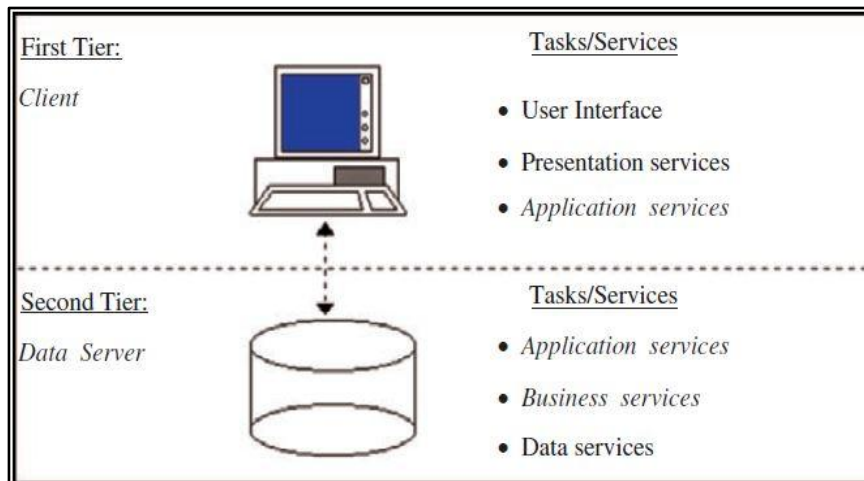
## Q. Explain about Database Architecture?

### Database Architecture

Database architecture essentially describes the location of all the pieces of information that make up the database application. The database architecture can be broadly classified into **Two-tier, Three-tier and Multitier architecture**.

### 1. Two-Tier Architecture

The two-tier architecture is a client–server architecture in which **the client** contains the presentation code and the SQL statements for data access. The database **server** processes the SQL statements and sends query results back to the client. The two-tier architecture is shown below.



### **Presentation Services**

“Presentation services” refers to the portion of the application which presents data to the user. In addition, it also provides for the mechanisms in which the user will interact with the data.. The presentation of the data should generally not contain any validation rules.

### **Business Services/objects**

“Business services” are a category of application services. Business services encapsulate an organizations business processes and requirements. These rules are derived from the steps necessary to carry out day-today business in an organization. These rules can be validation rules, used to be sure that the incoming information is of a valid type and format, or they can be process rules, which ensure that the proper business process is followed in order to complete an operation.

### **Application Services**

“Application services” provide other functions necessary for the application.

### **Data Services**

“Data services” provide access to data independent of their location. The data can come from legacy mainframe, SQL RDBMS,. Once again, the data services provide a standard interface for accessing data.

### **Advantages of Two-tier Architecture**

1. It is a good approach for systems with stable requirements and a moderate number of clients.
2. It is simple to implement, due to the number of good commercial development environments.

### **Drawbacks of Two-tier Architecture**

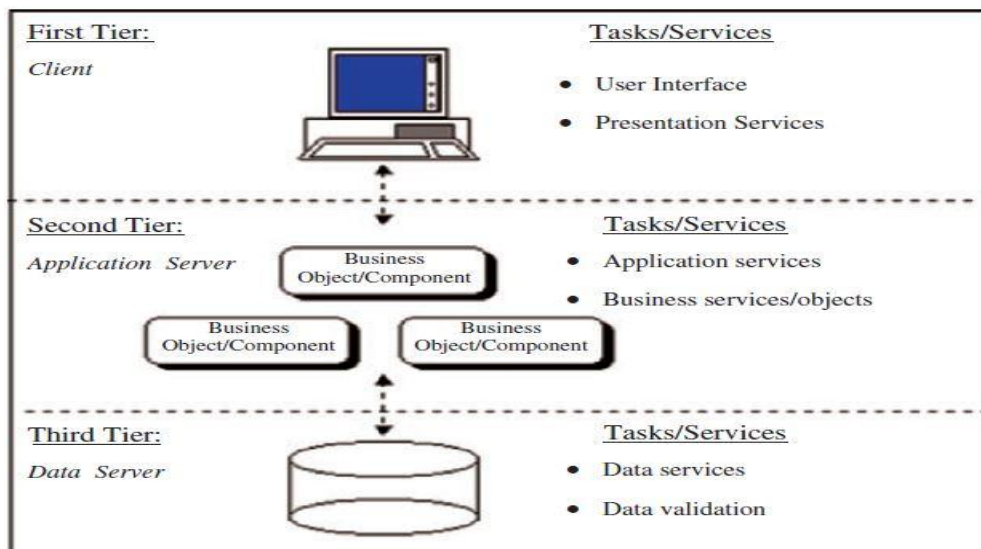
1. Software maintenance can be difficult
2. To make a significant change in the business logic, code must be modified on many PC clients.
3. The performance of two-tier architecture can be poor
4. For a large number of simultaneous clients, three-tier architecture may be necessary.

## 2. Three-tier Architecture

A three-tier architecture is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms. Three-tier architecture is a software design pattern and a well-established software architecture.

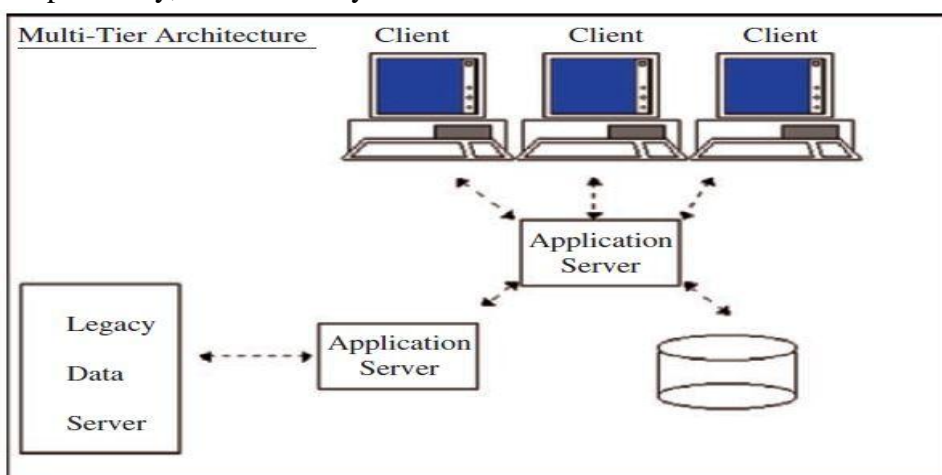
The three tiers in a three-tier architecture are:

1. **Presentation (User) Tier** – End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.
2. **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
3. **Data (Database) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.



## 3. Multitier Architecture

A multi-tier, three-tier, or N-tier implementation employs a three-tier logical architecture super imposed on a distributed physical model. Application Servers can access other application servers in order to supply services to the client application as well as to other Application Servers. The multiple-tier architecture is the most general client-server architecture. It can be most difficult to implement because of its generality. However, a good design and implementation of multiple-tier architecture can provide the most benefits in terms of scalability, interoperability, and flexibility.



## **Situations where DBMS is not Necessary**

### **DBMS is undesirable under following situations:**

- DBMS is undesirable if the application is simple, well-defined, and not expected to change.
- Runtime overheads are not feasible because of real-time requirements.
- Multiple accesses to data are not required.

Compared with file systems, databases have some disadvantages:

1. High cost of DBMS which includes:
  - Higher hardware costs
  - Higher programming costs
  - High conversion costs
2. Slower processing of some applications
3. Increased vulnerability
4. More difficult recovery

### **Data Dictionary**

A data dictionary, also known as a “system catalog,” is a centralized store of information about the database. It contains information about the tables, the fields the tables contain, data types, primary keys, indexes, the joins which have been established between those tables, referential integrity, cascades update, cascade delete, etc. This information stored in the data dictionary is called the “Metadata.”

### **Metadata**

The information (data) about the data in a database is called Metadata. The Metadata are available for query and manipulation, just as other data in the database.

# UNIT - II

## ENTITY-RELATIONSHIP MODEL

### Q. What are the Building blocks of a data model?

Basic building blocks of all data model are

1. **Entities:** Entity is any thing in which data to be collected. An entity is represented by a set of attributes. Eg. Name, ID, street, city for “customer” entity
2. **Attribute:** It is a characteristic of an entity. For example CUSTOMER entity has an attribute like customer name, address, phone number etc. It is used to represent fields in file system.
3. **Relation:** It describes the association between entities. There exist three types of relations. They are one-to-one, One-to-Many, Many-to-Many.
4. **Constraint:** It is a restriction placed on a data. It is used to ensure data integrity. They are expressed as rules like “Each class must have one and only teacher”.

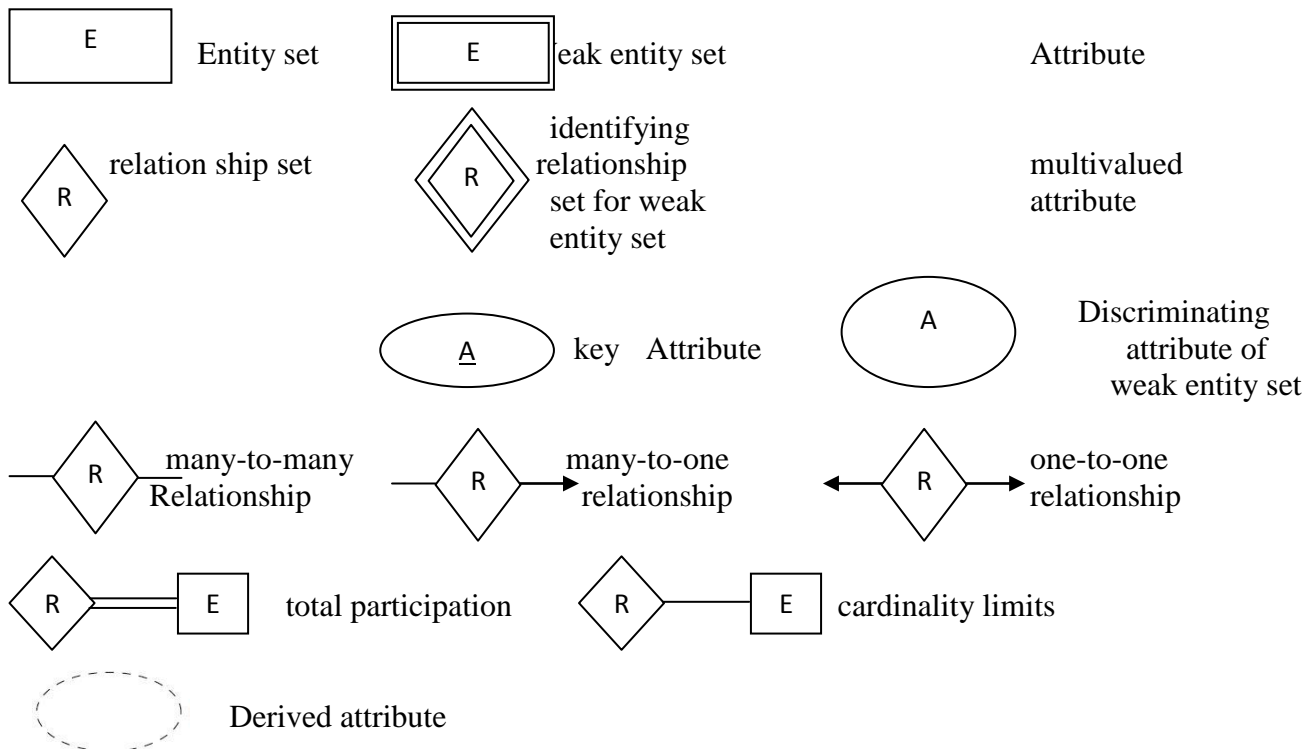
### Q. Explain ER model?

ER model is a high level conceptual data model development by Chen in 1976. It is used to represent entities, relations, attributes and constraints in graphical notation. The components (entity, attribute, relation) are represented by using the following symbols.

#### Symbols Used in ER Diagram

The elements in ER diagram are Entity, Attribute, and Relationship.

The components (entity, attribute, relation) are represented by using the following symbols.

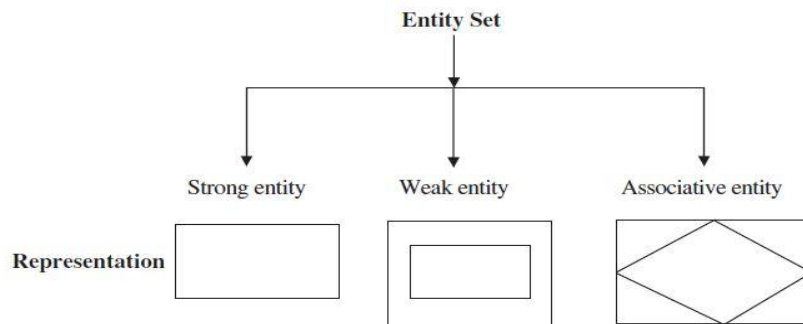


## Classification of Entity Sets

### **(Q. Explain about Classifications of Entity Sets?)**

Entity sets can be broadly classified into:

1. Strong entity.
2. Weak entity.
3. Associative entity.

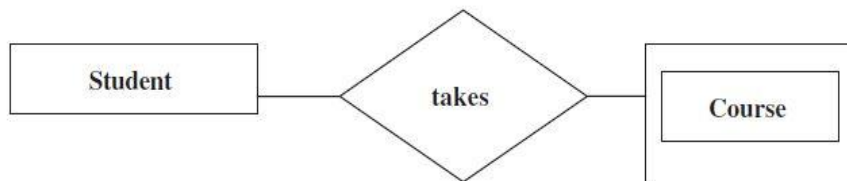


### **1. Strong entity set:**

A strong entity set has a primary key. All tuples in the set are distinguishable by that key.

#### **Example**

Consider the example, student takes course. Here student is a strong entity.



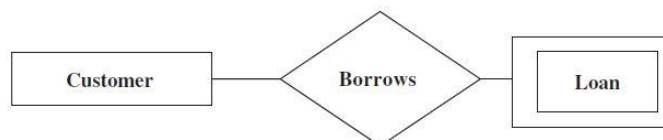
In this example, course is considered as weak entity because, if there are no students to take a particular course, then that course cannot be offered. The COURSE entity depends on the STUDENT entity.

### **2. Weak entity set:**

A Weak Entity is an entity that cannot be uniquely identified by its attributes alone, therefore, it must use a foreign key in conjunction with its attributes to create a primary key. The foreign key is typically a primary key of an entity it is related to.

#### **Example**

Consider the example, customer borrows loan. Here loan is a weak entity. For every loan, there should be at least one customer. Here the entity loan depends on the entity customer hence loan is a weak entity.



### **3. Associative entity:**

An **associative entity** is a term used in relational and entity-relationship theory. A relational database requires the implementation of a base relation (or base table) to resolve many-to-many relationships

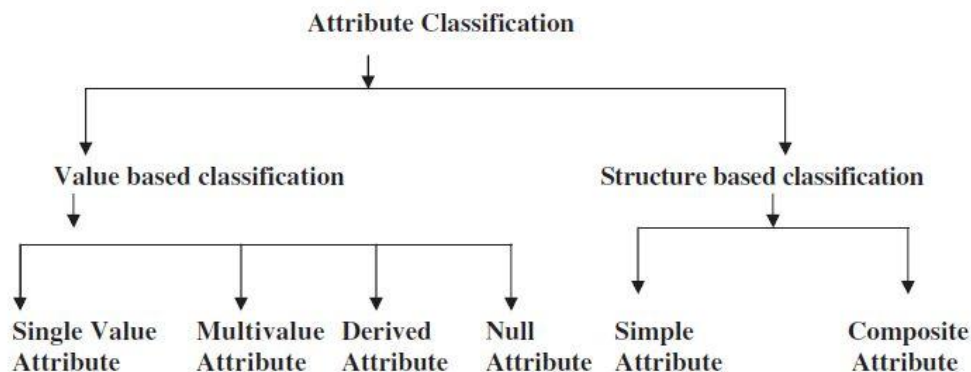




## Attribute Classification:

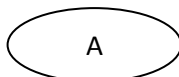
### **(Q. Explain about Classifications of Attributes?)**

Attribute is used to describe the properties of the entity. This attribute can be broadly classified based on value and structure. Based on value the attribute can be classified into single value, multi value, derived, and null value attribute. Based on structure, the attribute can be classified as simple and composite attribute.



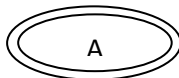
**Single-valued attribute:** It can take only a single value for each entity instance.

**Eg:** age of employee, Roll no of the student. There can be only one value for this.



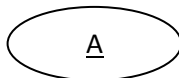
**Multi-valued attribute:** Attributes with a set of values for a particular entity.

**Eg:** address



**Key-attribute:** The attribute that is unique for every entity instance.

**Eg:** the account number of an account, the employee id of an employee etc.



**Composite attribute:** An attribute is a combination of two or more attributes is called composite attribute. It can be split into components.

**Eg:** Date of joining of the employee can be split into day, month and year.

**Stored attribute:** Attribute that need to be stored permanently.

**Eg:** name of an employee

**Derived attribute:** Attribute that can be calculated based on other attributes.

**Eg:** years of service of employee can be calculated from data of joining and current data.

**Null Value Attribute:** In some cases, a particular entity may not have any applicable value for an attribute. For such situation, a special value called null value is created.

**Domain of an attribute:** The set of possible values for an attribute is called the domain of the attribute. For example

1. The domain of attribute marital status is having four values: single, married, divorced or widowed.
2. The domain of the attribute month is having twelve values ranging from January to December.

## Relationship Degree

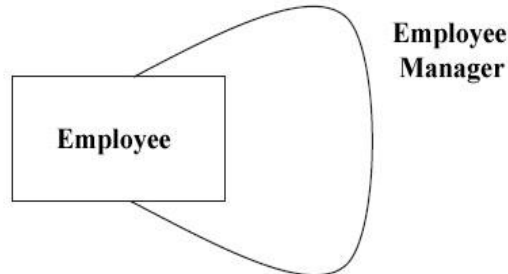
### (Q. Explain about Relationship Degree? )

Relationship degree refers to the number of associated entities. The relationship degree can be broadly classified into unary, binary, and ternary relationship.

#### 1. Unary Relationship

The unary relationship is otherwise known as recursive relationship. In the unary relationship the number of associated entity is one. An entity related to itself is known as recursive relationship.

**Ex:** employee manager of employee is unary



#### 2. Binary Relationship

In a binary relationship, two entities are involved.

**Ex:** each staff will be assigned to a particular department. Here the two entities are STAFF and DEPARTMENT.



#### 3. Ternary Relationship

In a ternary relationship, three entities are simultaneously involved.

**Ex:** customer purchase item, shop keeper is a ternary relationship



## Relationship Classification

### (Q. Explain about Classifications of Relationship? Or Explain about Association Tuples?)

Relationship is an association among one or more entities. This relationship can be broadly classified into **one-to-one** relation, **one-to-many** relation, **many-to-many** relation and recursive relation.

Relationship type	Representation	Example
One-to-one	↔	PRESIDENT ↔ COUNTRY
One-to-many	←	DEPARTMENT ← EMPLOYEES
Many-to-many	↔	EMPLOYEE ↔ PROJECT
Many-to-one	→	EMPLOYEE → DEPARTMENT

### 1. One-to-Many Relationship Type

The relationship that associates one entity to more than one entity is called one-to-many relationship.

**Ex:**one-to-many relationship is Country having states. For one country there can be more than one state hence it is an example of one-to-many relationship.

### 2. One-to-One Relationship Type

In a one-to-one relationship, one instance of one entity is associated with a single instance of another entity.

**Ex:** The relationship between the President and the country is an example of one-to-one relationship.

### 3. Many-to-Many Relationship Type

In many-to-many relationship many instances of one entity is associate with many instances of another entity.

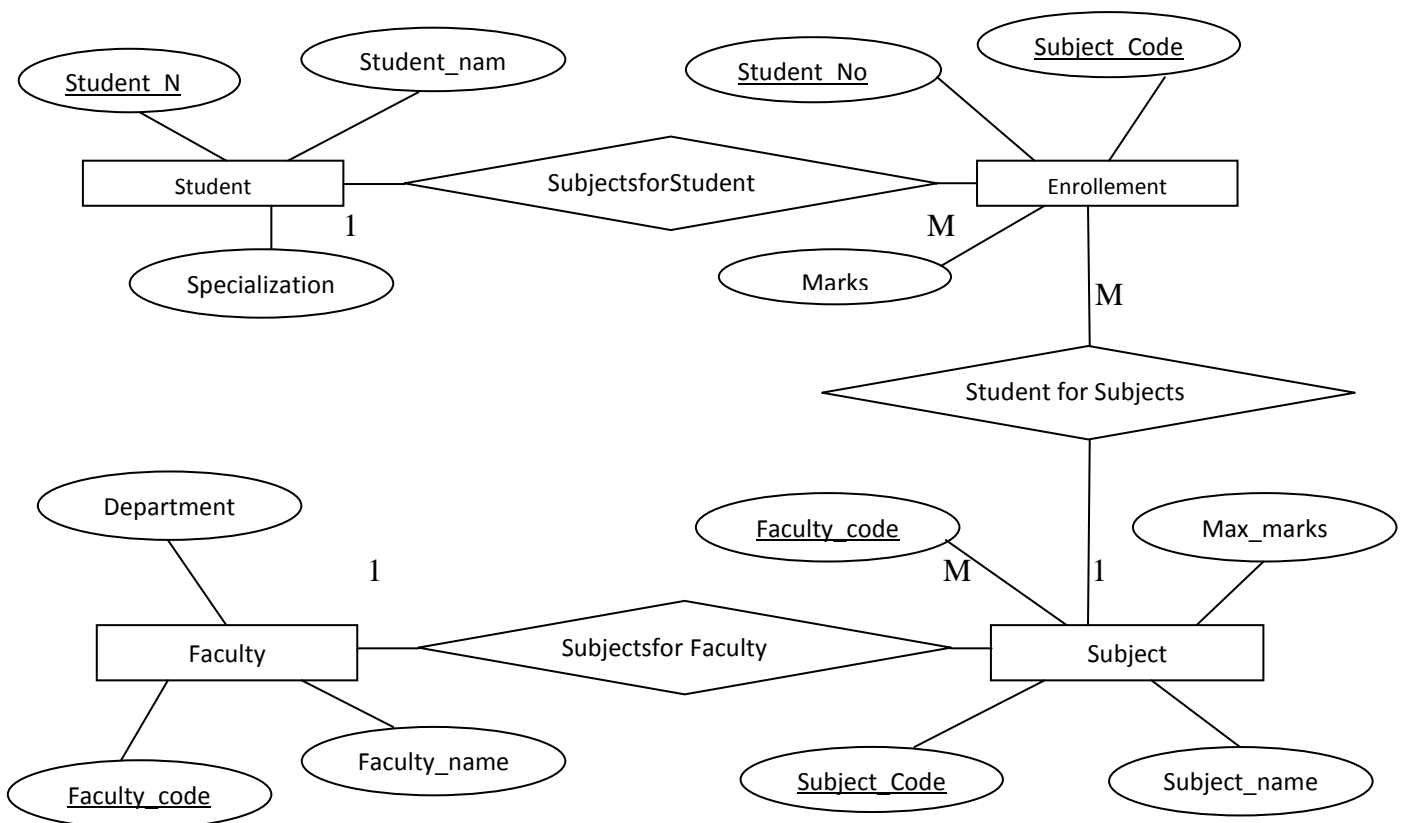
**Ex:**The relationship between EMPLOYEE entity and PROJECT entity. Many employees will be working in many projects hence the relationship between employee and project is many-to-many relationship.

### 4. Many-to-One Relationship Type

The relationship that associates more than one entity to one entity is called Many-to-One relationship.

**Ex:**The relationship between EMPLOYEE and DEPARTMENT. There may be many EMPLOYEES working in one DEPARTMENT.

**For example:** Construct an ER diagram for a college is as follows



## Extended Entity Relationship Model(EERM):

It is also called Enhanced Entity Relationship Model. This model is used to represent objects using Object Oriented Data Model. These diagrams represents the complexities of the data structure available in the tables. This model contains

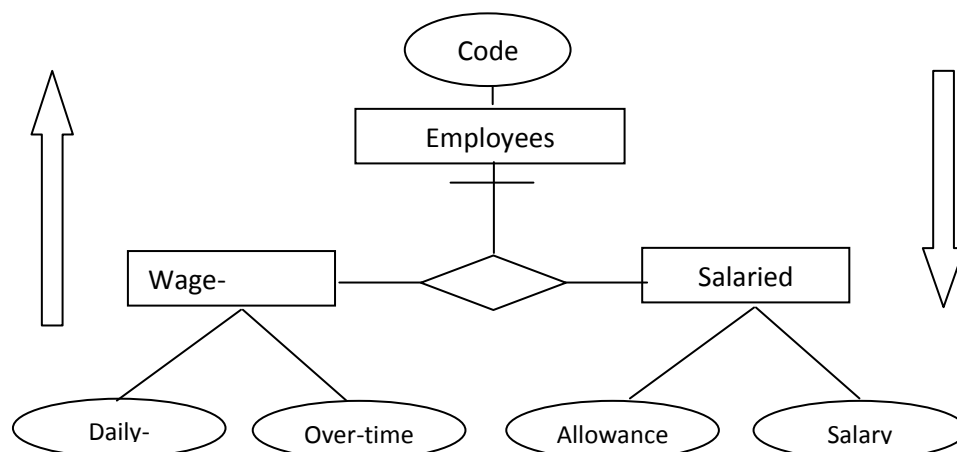
1. Entity Supertype and Subtypes
2. Specialization Hierarchy
3. Inheritance
4. Subtype Discriminator
5. Disjoint and Overlapping Constraints
6. Specialization and Generalization

### 1. Entity Supertype and Subtypes:

A subtype is a subset of another entity. For example, consider the entity Employee. There are two types of employees they are salaried employees and wage-earning employees. In this example, employee is a supertype and salaried and wage-earning employees are the subtypes. There would be some attributes such as 'name' and 'address' that are common to the subtypes. Wage-earning employees will have some attributes such as 'overtime' and 'daily wage' that do not belong to the salaried subtype. The salaried subtype has some attributes such as 'allowance' and 'salary' that do not belong to the subtype wage-earning.

Generalization

Specialization



A subentity (subtype) is always dependent on the superentity for its existence. The attributes of the superentity apply to all of its subentities. The subtype are connected to the supertype via an unnamed relationship. The supertype is connected to the relationship by a line containing a crossbar. The super type is described by the attributes that are unique to it, these are he attributes that do not belong to the other subtypes.

### 2. Specialization Hierarchy:

Entity supertypes and subtypes are organized in a specialization hierarchy, which depicts the arrangement of higher-level entity supertypes(Parent entity) and lower-level entity subtypes(Child entities). In the above diagram the specialization hierarchy formed by an employee supertype and two entities subtypes wage-earning and salaried. The specialization hierarchy reflects the 1:1 relationship between employee and its subtypes. For example a salaried subtype occurrence is related to one instance of the employee super types, and wage-earning subtype occurrence is related to one instance of the employee supertype.

The relationship represented within the specialization hierarchy are sometimes described in terms of 'is-a' relationships. For example a salaried is an employee, a wage-earning is an employee. It is important to understand that within a specialization hierarchy, a subtype can exist within the context of a supertype and every subtype can have only one supertype to which it is directly related.

### **3. Inheritance:**

The property of inheritance enables an entity subtype to inherit the attributes and relationships of the supertype. A supertype(base) contains those attributes that are common to all of its subtypes(derive). In contrast, subtypes contain only the attributes that are unique to the subtype.

### **4. Subtype Discriminator:**

A subtype Discriminator is the attribute in the supertype entity that determines to which subtype the supertype occurrence is related. It is common practice to represent the subtype discriminator and its value for each subtype in the ER diagram. However, not all ER modeling tools follow that practice.

### **5. Disjoint and Overlapping Constraint:**

An entity supertype can have disjoint and overlapping entity subtypes. A disjoint subtype contain a unique subset of the supertype entity set, in other words, each entity instance of the supertype can appear in only one of the subtypes.

Overlapping subtype represents that supertype is part of multiple classifications, the employee supertype may contain overlapping job classification subtypes. It means an employee belongs to both salaried and wage-earning.

### **6. Specialization and Generalization:**

Specialization is the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping unique characteristics and relationships of the subtypes. In the employee example, you used specialization to identify multiple entity subtypes from the original employee supertype. Generalization is the bottom-up process of identifying a higher-level, more generic entity supertype from lower-level entity subtypes. Generalization is based on grouping common characteristics and relationships of the subtypes.

### **Entity Clustering:**

Developing ER diagram requires hundreds of entity types and their respective relationships that crowd the diagram to the point of making it unreadable and inefficient as a communication tool. In those cases, we can use entity clusters to minimize the number of entities.

An entity cluster is a “virtual” entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities into a single abstract entity object.

### **Advantages of ER Modelling:**

An ER model is derived from business specifications. ER models separate the information required by a business from the activities performed within a business. Although business can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant. The advantages of ER are

1. The ER modeling provides an easily understood pictorial map for the database design.
2. It is possible to represent the real world problems in a better manner in ER modelling.
3. The conversion of ER model to relational model is straightforward.
4. The enhanced ER model provides more flexibility in modeling real world problems.
5. The symbols used to represent entity and relationships between entities are simple and easy to follow.

# UNIT - III

## RELATIONAL DATA MODEL

### Q.What are CODD's Rules?

In 1985, Codd published a list of rules that became a standard way of evaluating a relational system. After publishing the original article Codd stated that there are no systems that will satisfy every rule. Nevertheless the rules represent relational ideal and remain a goal for relational database designers.

0. **Foundation Rule:** A relational database management system must manage its stored data using only its relational capabilities.
1. **Information Rule:** All information in the database should be represented in one and only one way- by values in a table.
  - Data should be presented to the user in the tabular form.
2. **Guaranteed Access Rule:** Each and every datum(atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.
  - Every data element should be unambiguously accessible.
3. **Systematic Treatment of Null Values:** Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.
4. **Dynamic On-line Catalog Based on the Relational Model:** The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.
  - The database description should be accessible to the users.
5. **Comprehensive Data Sublanguage Rule:** A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible: data definition, view definition, data manipulation(interactive and by program)integrity constraints, authorization and transaction boundaries(begin, commit, and rollback)
  - A database supports a clearly defined language to define the database, view the definition, manipulate the data, and restrict some data values to maintain integrity.
6. **View Updating Rule:** All views that are theoretically updateable are also updateable by the system.
  - Data should be able to be changed through any view available to the user.
7. **High-level Insert, Update and Delete:** The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.
  - All records in a file must be able to be added, deleted, or updated with singular commands
8. **Physical Data Independence:** Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.
  - Changes in how data are stored or retrieved should not affect how a user accesses the data.
9. **Logical Data Independence:** Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.
  - A user's view of data should be unaffected by its actual form in files.
10. **Integrity Independence:** Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.
  - Constraints on user input should exist to maintain data integrity.
11. **Distribution Independence:** The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.
  - A database design should allow for distribution of data over several computer sites.
12. **Non subversion Rule:** If a relational system supports a low-level(single-record-at-a-time) language, that low-level language cannot be used to bypass the integrity rules expressed in the higher-level(multiple-records-at-a-time) relational language.
  - Data fields that affect the organization of the database cannot be changed.

## Table & properties of a table

It can be represented as a matrix of rows and columns. Each row represents record or tuple and each column represent an attribute or field. Each row/column intersection represents a single data value. All values in a column has a specific range of values knows as the attribute domain. The order of the rows and columns is immaterial to the DBMS. This relation has the following attributes.

1. Each column of the relation contains values about the same attribute of the entity for which the relation is defined
2. Each cell value in the relation must be simple
3. Each column has a distinct name and the ordering columns is immaterial
4. Each row is distinct
5. The order of the rows of the relation is immaterial

### Super key

An attribute or combination of attributes that uniquely identifies each row in a table

### Candidate key

It is a minimal super key. A super key that does not contain a subset of attributes that is itself a super key.

### Primary Key

A primary key is a key which uniquely identifies all the row of a table. This key does not allow duplicate values and null values. In a table there exist only one primary key. For example

Empno	Empname	Salary
1	Vinay	40000
2	Sunny	35000
3	Bhanu	38000

### Secondary key

An attribute or combination of attributes used to retrieve data from a table

### Foreign key

Attribute or combination of attributes in one table whose values must either match the primary key in another table or be null.

### Integrity rules

There are two types of integrity rules they are

1. Entity integrity
2. Referential integrity

1. **Entity integrity** means all primary key entries are unique and not null. For each row will have a unique identity, and foreign key values can properly reference primary key values.

Ex: All invoices are uniquely identified by their invoice number”

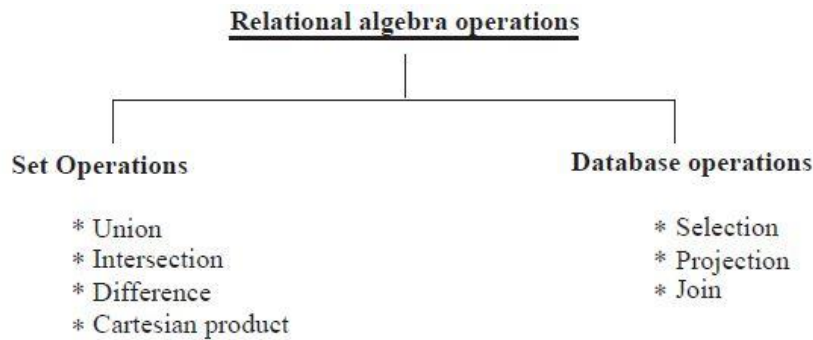
2. **Referential integrity** means every non-null foreign key value must reference an existing primary key value. It is not possible to insert new row in a table with out having matching primary key on other table. It is also used to delete all foreign key records of a table if you delete a primary key on another table(on delete cascade).

Ex: A customer might not yet have an assigned sales representative number but it will be impossible to have an invalid sales representative number.



## Relational algebra operators

In RDBMS the files are stored as relations (tables). Between relations there exist several relationships. These relationships in RDBMS can be classified into **relational algebra** and **relational calculus**.



## Relational Algebra

The relational algebra contains two types of operator's namely

### 1. Traditional Set operators:

The traditional set operators are union, intersection, difference and Cartesian product.

### 2. Special relational operators:

The special relational operators are selection, projection, join and division

For example :

Table A

Sno	Name
1	Rama
2	Sita
3	Raju

Table B

Sno	Name
1	Rama
2	Satya
4	Rani

### 1. Traditional set operators:

**Union:** The union of two compatible relations A and B is a new relation which contains set of records belongs to A, B both A and B. It is possible to apply union if both the relations A and B having same number of columns

Example:  $A \cup B$ , A UNION B gives

Table A union Table B

Sno	Name
1	Rama
2	Sita
2	Satya
3	Raju
4	Rani

**Intersection:** the intersection of two compatible relations A and B is a new relation which consists of all rows which are common to both A and B.

Example: A intersection B

Sno	Name
1	Rama

**Difference:** It results all rows in one table that are not found in another table, i.e it subtract one table from other.

A-B

Sno	Name
2	Sita
3	Raju

**Cartesian product:** It is used to join every row of one table to every row of another table. If the relation A has m rows and the relation B has n rows then the number of rows in the resultant relation is mXn.

Ex: A times B

### Special relational operators:

The special relational operators are selection, projection, join and division

**Selection:** The selection operator applied on a relation creates new relation that contains horizontal subset of records which satisfies a certain condition. It contains retrieved horizontal subset of records. It is represented by using operator is sigma  $\sigma$ . It can written as  $\sigma C^{\circledast}$  where R is relation and C is condition.

Example Select only those Employees Name is Rama.  $\sigma \text{Name='Rama'}(A)$  produce

Sno	Name
1	Rama

**Projection:** The projection operator applied on a relation creates new relation that contains vertical subset of records which satisfies a certain condition. It eliminated the duplicate records with in the selected attributes. It can be represented by using the operator pi  $\pi$ . It can be written as  $\pi_{\text{attributes}} R$  where attributes is the list of attributes to be displayed and R is the relation.

For example Project only the names of table A:  $\pi_{\text{name}} (A)$  produce

Name
Rama
Sita
Raju

**Join:** Join operator is used to combine two relations A and B based on values of a common attribute for both the relations to form a new relation C. It can be represented by using the symbol ---.

For example display sno, name from table A, table B where table A.sno=tableB.sno

Sno	Name
1	Rama

The join condition can be =  $\diamond$   $\geq$   $\leq$   $\neq$


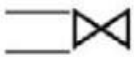
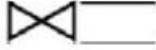
If the join condition is = it is called equijoin or inner join or join and if the join condition is other than = it is called non equijoin or theta join.

**Natural join:** A natural join compares the common columns of both ( A, B) tables with each other and produce a resultant relation C. The resultant relation C does not contains duplicate values. It can be represented by using the operator \* . In natural join the join condition is = between two common attributes.

Example: Emp\*Depart

## Outer join:

An outer join is a join statement that uses an = operation to match rows from different tables. It displays equal rows from both table and non equal record from both the tables. Three types of outer joins:

1. Full outer join 
2. Left outer join 
3. Right outer join 

**Full Outer Join:** includes all records in the left hand relation and right hand relation.

Sno	Name	Sno	Name
1	Rama	1	Rama
2	Sita	2	Satya
3	Raju	Null	Null
Null	Null	4	Rani

**Left Outer Join:** includes all rows in the left hand relation and includes only those matching rows from the right hand relation.

Sno	Name	Sno	Name
1	Rama	1	Rama
2	Sita	2	Satya
3	Raju	Null	Null

**Right Outer Join:** includes all records in the right hand relation and includes only those matching records from the left hand relation.

Sno	Name	Sno	Name
1	Rama	1	Rama
2	Sita	2	Satya
Null	Null	4	Rani

**Self Join:** A self join is a join in which a table is joined with itself.

## Division Operation

The division of the relation R by the relation S is denoted by  $R \div S$

To illustrate division operations consider two relations **STUDENT** and **MARK**. The **STUDENT** relation has the attributes Student Name and the mark in particular subject say mathematics. The **MARK** relation consists of only one column mark and only one row.

Student		Mark
Name	Mark	Mark
Arul	97	100
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

If we divide the STUDENT relation by the MARK relation, the resultant relation is shown as:

<u>Answer</u>
<u>Name</u>
Banu
Dinesh

## **Advantages of Relational Algebra**

The relational algebra has solid mathematical background. The mathematical background of relational algebra is the basis of many interesting developments and theorems. If we have two expressions for the same operation and if the expressions are proved to be equivalent, then a query optimizer can automatically substitute the more efficient form. Moreover, the relational algebra is a high level language which talks in terms of properties of sets of tuples and not in terms of for-loops.

## **Limitations of Relational Algebra**

The relational algebra cannot do arithmetic. For example, if we want to know the price of 10 l of petrol, by assuming a 10% increase in the price of the petrol, which cannot be done using relational algebra. The relational algebra cannot sort or print results in various formats. For example we want to arrange the product name in the increasing order of their price. It cannot be done using relational algebra. Relational algebra cannot perform aggregates. For example we want to know how many staff are working in a particular department. This query cannot be performed using relational algebra.

## **Data dictionary and System catalog**

**Data dictionary** is used to provide a detailed accounting of all tables found within the user create database. It contains all the attribute names and characteristics for each table in the system. It is also called the database designers database.

**System catalog** contains metadata. It can be described as a very detailed system data dictionary that describes all objects within the database including data about table names, tables creator and creation date, the number of columns in each table the data type corresponding to each column, index file names index creators, authorized users, access privileges etc. it is a system created database. It contains user created characteristics and contents.

## **Homonym**

Homonym is used to indicate the use of same attribute name to label different attributes. For example c\_name is used for customer name or consultant name.

## **Synonym**

Synonym is used to indicate the use of different names to describe the same attribute. For example car and auto refer to the same object.

## **Index**

An index is orderly arrangement used to logically access rows in a table. It is an ordered arrangement of key, pointers and each key point to the location of the data identified by the key. Index is used to retrieve data from a table, sort the data on a field. For example creating an index on customer last name will allow you to retrieve the customer data alphabetically by customer last name. Index also used to implement primary keys. When you define a primary key the DBMS automatically creates a unique index on the primary key column. A table can have many indexes, but each index is associated with only one table. The index key can have multiple attributes.

## UNIT IV – Structured Query Language

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as discussed below:

### **1. DDL(Data Definition Language) :**

These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.

#### **Examples of DDL commands:**

- **Create**
- **Alter**
- **Drop**
- **Truncate**
- **Rename**

#### ❖ **CREATE:**

It is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

##### **Syntax:**

```
Create table tablename(colname1 datatype(size),
                        colname2 datatype(size),
                        -----
                        colnameN datatype(size));
```

##### **Ex:**

```
Create table stu(rno number(4),
                sname varchar2(20),
                maths number(3),
                physics number(3),
                computer number(3));
```

#### ❖ **Alter:**

Alter command is used to add a new column to a table and modify an existing column

\* Note that changing of datatype of a column and decreasing the size of a column. It is possible if and only if the column is empty.

#### **Add a new column:**

##### **Syntax:**

```
Alter table tablename add(cname datatype(size));
```

##### **Ex:**

```
Alter table stu add(total number(3));
```

#### **Modify an existing column:**

##### **Syntax:**

```
Alter table tablename modify(cname newdatatype(newsize));
```

##### **Ex:**

```
Alter table stu modify(sname varchar2(20));
Alter table stu modify(rno number(4));
```

#### ❖ **DROP**

This command is used to delete all the rows in the table along with the table structure.

##### **Syntax:**

```
Drop table tablename;
```

##### **Ex:**

```
Drop table stu;
```

## ❖ **TRUNCATE**

It is used to remove all records from a table, including all spaces allocated for the records are removed.

### Syntax:

Truncate table tablename;

### Ex:

Truncate table stu;

## ❖ **Rename**

### Syntax:

Rename table old\_tablename To new\_tablename;

### Ex:

Rename table stu to student;

## 2. **DML(Data Manipulation Language) :**

These SQL commands are used for storing, retrieving, modifying, and deleting data. These Data Manipulation Language commands are: [SELECT](#), [INSERT](#), [UPDATE](#), and [DELETE](#).

- **SELECT** – is used to retrieve data from the a database.
- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

## ❖ **SELECT:**

The most commonly used SQL command is Select statement. This command is used to Query or retrieve data from a table in the database. A Query may retrieve information from specified columns or from all of the columns in the table.

- i. All rows and All columns
- ii. All rows and Selected columns(**Projection Operation**)
- iii. Selected rows and selected columns
- iv. Selected rows and all columns(**Selection Operation**)

### i) All rows and All columns:

#### Syntax:

select \* from tablename;

#### Ex:

select \* from stu;

### ii) All rows and Selected columns: (Projection Operation)

The projection operation performs column wise filtering. Specific columns are selected in projection operation.

#### Syntax:

select cname1,cname2,...cnamen from tablename;

#### Ex:

select rno,sname from stu;

### iii) Selected rows and Selected columns:

#### Syntax:

select cname1,cname2,...cnamen from tablename where condition;

#### Ex:

select rno,sname from stu where total>360;

### iv) Selected rows and All columns: (Selection Operation)

Selection operation can be considered as row wise filtering. We can select specific rows using condition.

#### Syntax:

select \* from tablename where condition;

#### Ex:

select \* stu where maths>75;

## ❖ INSERT

- **This command is used to add new rows of data to a table.**

### Syntax:

Insert into tablename values(value1,value2,.....,valuen);

### Ex:

Insert into stu values(100,'raj',85,90,92);

- **To enter multiple rows into the table**

### Syntax:

Insert into tablename values(&cname1,&cname2,...,&cnamen);

### Ex:

Insert into stu values(&rno,'&sname',&maths,&physics,&computers);

## ❖ UPDATE

This command is used to calculate the data values in a table by using this command we can update all the rows from a table or selected rows from the table.

### To update all the rows:

#### Syntax:

Update tablename set cname1=exp1,cname2=exp2;

#### Ex:

Update stu set total=maths+physics+computers,average=total/3;

### To update specific rows:

#### Syntax:

Update tablename set cname1=exp1 where condition;

#### Ex:

Update stu set math=95 where rno=2051;

## ❖ DELETE

This command is used to remove the rows from the table. We can delete the rows in two ways

- i) Removing all rows
- ii) Removing specific rows

### i) Removing all rows:

#### Syntax:

Delete from tablename;

#### Ex:

SQL>Delete from stu;

### ii) Removing specific rows:

#### Syntax:

Delete from tablename where condition;

#### Ex:

SQL>Delete from stu where math<35;

## 3. **DCL(Data Control Language) :**

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

### **Examples of DCL commands:**

- **GRANT**-gives user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.



#### 4. **TCL(transaction Control Language) :**

TCL commands deals with the transaction within the database.

##### **Examples of TCL commands:**

- **COMMIT**– commits a Transaction.

**Syntax:**

Commit;

- **ROLLBACK**– rollbacks a transaction in case of any error occurs.

**Syntax:**

Roll back;

- **SAVEPOINT**–sets a savepoint within a transaction.

**Syntax:**

Savepoint A;

## SQL General Data Types:

Each column in a database table is required to have a name and a data type.

SQL developers have to decide what types of data will be stored inside each and every table column when creating a SQL table. The data type is a label and a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

The following table lists the general data types in SQL:

<b>Data type</b>	<b>Description</b>
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER or INT	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

# SQL CONSTRAINTS

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

## SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

### Syntax:

```
CREATE TABLE table_name (  
    column1 datatype constraint,  
    column2 datatype constraint,  
    column3 datatype constraint,  
    ....  
);
```

The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition

❖ **NOT NULL Constraint** : Ensures that a column cannot have a NULL value

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);
```

❖ **UNIQUE Constraint**

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE constraint.

- **UNIQUE Constraint on CREATE TABLE**

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);
```

- **UNIQUE Constraint on ALTER TABLE**

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons ADD UNIQUE (ID);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

- **UNIQUE Constraint on DROP**

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE Persons DROP CONSTRAINT UC_Person;
```

## ❖ **PRIMARY KEY Constraint**

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

- **PRIMARY KEY on CREATE TABLE**

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (  
  ID int NOT NULL PRIMARY KEY,  
  LastName varchar(255) NOT NULL,  
  FirstName varchar(255),  
  Age int  
);
```

- **PRIMARY KEY on ALTER TABLE**

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons ADD PRIMARY KEY (ID);
```

- **PRIMARY KEY on DROPTABLE**

To drop a PRIMARY KEY constraint, use the following SQL:

```
ALTER TABLE Persons DROP PRIMARY KEY;
```

## ❖ FOREIGN KEY Constraint:

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

"Persons" table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

- **FOREIGN KEY on CREATE TABLE**

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,  
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)  
);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);
```

- **FOREIGN KEY on ALTER TABLE**

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

```
ALTER TABLE Orders ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

- **DROP a FOREIGN KEY Constraint**

To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE Orders DROP CONSTRAINT FK_PersonOrder;
```

## ❖ **CHECK Constraint**

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

- **CHECK on CREATE TABLE**

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that you can not have any person below 18 years:

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int CHECK (Age>=18)  
);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255),  
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')  
);
```

- **CHECK on ALTER TABLE**

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons ADD CHECK (Age>=18);
```

- **DROP a CHECK Constraint**

To drop a CHECK constraint, use the following SQL:

```
ALTER TABLE Persons DROP CONSTRAINT CHK_PersonAge;
```

## **SQL JOIN**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

<u>OrderID</u>	<u>CustomerID</u>	<u>OrderDate</u>
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

<u>CustomerID</u>	<u>CustomerName</u>	<u>ContactName</u>	<u>Country</u>
1	<u>Alfreds Futterkiste</u>	Maria Anders	Germany
2	<u>Ana Trujillo Emparedados y helados</u>	Ana Trujillo	Mexico
3	<u>Antonio Moreno Taqueria</u>	Antonio Moreno	Mexico

Then, look at a selection from the "Customers" table:

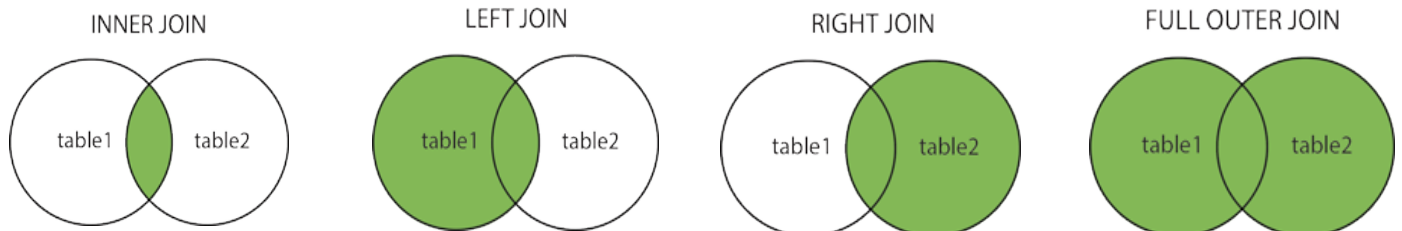
Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

## Different Types of JOINS

Here are the different types of the JOINS in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



### ❖ INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

#### **Syntax**

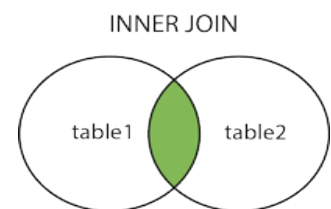
```
SELECT column_name(s) FROM table1  
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

The following SQL statement selects all orders with customer information:

#### **Example**

```
SELECT Orders.OrderID, Customers.CustomerName FROM Orders;
```

**Note:** The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not show!



### ❖ LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

#### **Syntax**

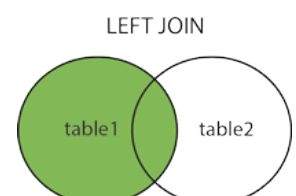
```
SELECT column_name(s) FROM table1 LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

**Note:** In some databases LEFT JOIN is called LEFT OUTER JOIN.

The following SQL statement will select all customers, and any orders they might have:

#### **Example**

```
SELECT Customers.CustomerName, Orders.OrderID FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;
```





**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

### ❖ RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

#### Syntax

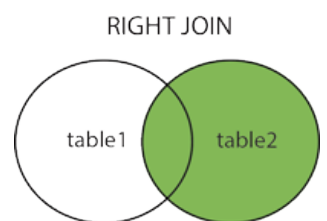
```
SELECT column_name(s) FROM table1  
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

**Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

The following SQL statement will return all employees, and any orders they might have placed:

#### Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders RIGHT JOIN Employees ON  
Orders.EmployeeID = Employees.EmployeeID ORDER BY Orders.OrderID;
```



**Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

### ❖ FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.

**Note:** FULL OUTER JOIN can potentially return very large result-sets!

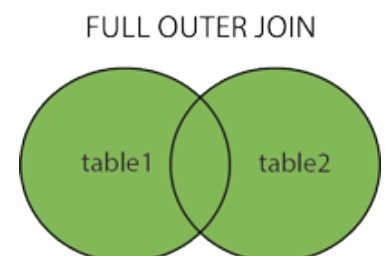
#### Syntax

```
SELECT column_name(s) FROM table1  
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```

#### Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers FULL OUTER JOIN Orders ON  
Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;
```



**Note:** The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

## ❖ Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

### Syntax

SELECT column\_name(s) FROM table1 T1, table1 T2 WHERE condition;

The following SQL statement matches customers that are from the same city:

### Example

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B WHERE A.CustomerID <> B.CustomerID AND A.City = B.City
ORDER BY A.City;
```

## Set Operation in SQL

SQL supports few Set operations to be performed on table data. These are used to get meaningful results from data, under different special conditions.

First		Second	
Sno	Name	Sno	Name
1	Rama	1	Rama
2	Sita	2	Satya
3	Raju	4	Rani

## Union

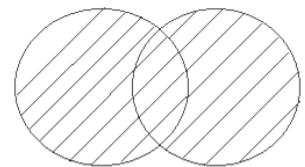
UNION is used to combine the results of two or more Select statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables.

### **Example of UNION**

Union SQL query will be,

```
select * from First UNION select * from second;
```

Sno	Name
1	Rama
2	Sita
2	Satya
3	Raju
4	Rani



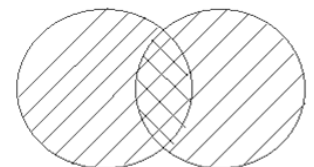
## Union All

This operation is similar to Union. But it also shows the duplicate rows.

### **Example of Union All**

```
select * from First UNION ALL select * from second;
```

Sno	Name
1	Rama
2	Sita
3	Raju
1	Rama
2	Satya
4	Rani



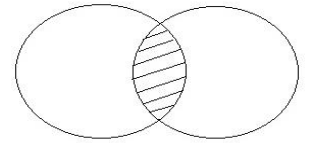
## Intersect

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same. MySQL does not support INTERSECT operator.

### Example of Intersect

select \* from First **INTERSECT** select \* from second;

Sno	Name
1	Rama



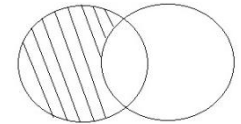
## Minus

Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support INTERSECT operator.

### Example of Minus

select \* from First **MINUS** select \* from second;

Sno	Name
2	Sita
3	Raju



## SQL aggregate functions

An aggregate function allows you to perform a calculation on a set of values to return a single scalar value. We often use aggregate functions with the **GROUP BY** and **HAVING** clauses of the **SELECT** statement. The following are the most commonly used SQL aggregate functions:

- **AVG** – calculates the average of a set of values.
- **COUNT** – counts rows in a specified table or view.
- **MIN** – gets the minimum value in a set of values.
- **MAX** – gets the maximum value in a set of values.
- **SUM** – calculates the sum of values.

- **SQL aggregate functions syntax:**

Selectt aggregate\_function (DISTINCT | ALL expression) from Tablename;

- First, specify an aggregate function that you want to use e.g., MIN, MAX, AVG, SUM or COUNT.
- Second, put DISTINCT or ALL modifier followed by an expression inside parentheses. If you explicitly use DISTINCT modifier, the aggregate function ignores duplicate values and only consider the unique values. If you use the ALL modifier, the aggregate function uses all values for calculation or evaluation. The ALL modifier is used by default if you do not specify any modifier explicitly.

**Table Emp**

<u>Eno</u>	<u>Ename</u>	<u>Job</u>	<u>Salary</u>
2001	A	Salesman	15,000
2002	B	Manager	30,000
2003	c	Salesman	17,000
2004	D	Manager	35,000

1. **AVG** – calculates the average of a set of values.

**Write a query to display average salary of employee whose commission is null**

```
select avg(salary) from emp where job='Manager';
      AVG(SALARY)
      -----
      16,000
```

2. **COUNT** – counts rows in a specified table or view.

Write a query to count the employee where salary is greater than 1500.

```
select count(eno) from emp;
COUNT(ENO)
-----
4
```

3. **MIN** – gets the minimum value in a set of values.

Write a query to display who draw the min salary in employee table where deptno is 20.

```
select min(salary) from emp where job='salesman';
MIN(SALARY)
-----
15,000
```

4. **MAX** – gets the maximum value in a set of values.

Write a query to display max salary of employee whose job is manager.

```
select max(salary) from emp where job='MANAGER';
MAX(SALARY)
-----
35,000
```

5. **SUM** – calculates the sum of values.

Write a query to calculate sum of salaries of the employee.

```
select sum(salary) from emp;
SUM (SALARY)
-----
92,000
```

### **GROUP BY Clause:**

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

Important Points:

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

**Syntax:**

```
SELECT attribute name, aggregate function FROM table name
GROUP BY attribute name;
Placement
```

Companyname	Department	Strength
TCS	CSE	54
TCS	ECE	40
TCS	EEE	32
GE	CSE	5
GE	ECE	8
GE	EEE	20
L&T	CSE	12
L&T	ECE	20
L&T	EEE	18
IBM	CSE	24
IBM	ECE	20

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select departmentname, sum(strength)
2 from placement
3 group by departmentname;
DEPARTMENTNAME      SUM(STRENGTH)
-----
CSE                   95
ECE                   88
EEE                   82
SQL> |

```

Attribute name after SELECT command should match with the attribute name (departmentname) after GROUP BY command.

**HAVING Clause:**

The HAVING command is used to select the group. In other words HAVING restricts the groups according to a specified condition.

**Syntax :**

**SELECT** attribute name, aggregate function  
**FROM** table name  
**GROUP BY** attribute name  
**HAVING** condition;

**Example:**

```

SQL> select departmentname, sum(strength)
2 from placement
3 group by departmentname
4 having sum(strength)>90;
DEPARTMENTNAME      SUM(STRENGTH)
-----
CSE                   95

```

**ORDER BY clause:**

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

**Syntax:**

**SELECT** column-list **FROM** table\_name [**WHERE** condition]  
[**ORDER BY** column1, column2, .. columnN] [**ASC** | **DESC**];

**Example:**

Select \* From placement **ORDER BY** department **DESC**;

Companyname	Department	Strength
TCS	EEE	32
GE	EEE	20
L&T	EEE	18
TCS	ECE	40
GE	ECE	8
L&T	ECE	20
IBM	ECE	20
TCS	CSE	54
GE	CSE	5
L&T	CSE	12
BM	CSE	24

## SQL Sequence

Sequence is a feature supported by some database systems to produce unique values on demand. Some DBMS like **MySQL** supports **AUTO\_INCREMENT** in place of Sequence. **AUTO\_INCREMENT** is applied on columns, it automatically increments the column value by 1 each time a new record is entered into the table. Sequence is also somewhat similar to **AUTO\_INCREMENT** but it has some extra features.

### Creating Sequence:

#### Syntax :

```
CREATE Sequence sequence-name
start with initial-value
increment by increment-value
maxvalue maximum-value
cycle|nocycle;
```

- The **initial-value** specifies the starting value for the Sequence.
- The **increment-value** is the value by which sequence will be incremented.
- The **maximum-value** specifies the upper limit or the maximum value upto which sequence will increment itself.
- The keyword **CYCLE** specifies that if the maximum value exceeds the set limit, sequence will restart its cycle from the beginning.
- And, **NO CYCLE** specifies that if sequence exceeds **MAXVALUE** value, an error will be thrown.

### Example to create Sequence:

The sequence query is following

```
CREATE Sequence sample start with 1 increment by 1 maxvalue 999 cycle ;
```

### Example to use sequence :

**create a table named students with columns as id and name.**

```
CREATE table students
(
ID number(10),
Name varchar2(20),
);
```

**Now insert values into table**

```
INSERT into students VALUES(sequence_1.nextval,'Rajesh');
INSERT into students VALUES(sequence_1.nextval,'Ambedkar');
```

**Output:**

ID	NAME
1	Rajesh
2	Ambedkar

## SQL Views

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

### Syntax :

```
CREATE view view_name AS SELECT column_name(s) FROM table_name WHERE condition;
```

SID	Name	Address
1	Rama	RJY
2	Sita	KKD
3	Raju	AMP

ID	Name	Marks	Age
1	Rama	90	20
2	Sita	85	19
3	Raju	80	19

### Creating View from a single table:

#### Syntax:

```
CREATE VIEW view_name AS  
SELECT column1, column2.....  
FROM table_name  
WHERE condition;
```

- **view\_name:** Name for the View
- **table\_name:** Name of the table
- **condition:** Condition to select rows

In this example we will create a View named DetailsView from the table StudentDetails.

#### Example:

```
CREATE VIEW DetailsView AS SELECT NAME, ADDRESS FROM StudentDetails WHERE SID < 5;
```

```
SELECT * FROM DetailsView;
```

#### Out put:

Sno	Name
1	Rama
2	Sita
3	Raju

### Creating View from Multiple tables:

#### Syntax:

```
CREATE VIEW View_name AS  
SELECT table1_name.colname1, table1_name.colname2,.. table2_name.colname1, table2_name.colname2...  
FROM table1_name, table2_name  
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

### Example:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

```
SELECT * FROM MarksView;
```

### Out put:

Name	Address	Marks
Rama	RJY	90
Sita	KKD	85
Raju	AMP	80

### Force View Creation:

Force keyword is used while creating a view. This keyword force to create View even if the table does not exist. After creating a force View, if we create the **base table** and enter **values** in it, the view will be **automatically updated**.

### Syntax:

```
CREATE force view view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

### Update a View:

Update command for view is same as for tables.

### Syntax :

```
UPDATE view-name set value WHERE condition;
```

If we update a view it also updates base table data automatically.

### Read-Only View:

We can create a view with read-only option to restrict access to the view.

### Syntax :

```
CREATE force view view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition with read-only
```

The above syntax will create view for read-only purpose, we cannot Update or Insert data into read-only view. It will throw an error.



# UNIT V

## PL/SQL

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.

### Features of PL/SQL

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

### Advantages of PL/SQL

- PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control. In Dynamic SQL allows embedding DDL statements.
- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages

### **PL/SQL structured**

S.No	Sections & Description
1	<b>Declarations</b> This section starts with the keyword <b>DECLARE</b> . It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	<b>Executable Commands</b> This section is enclosed between the keywords <b>BEGIN</b> and <b>END</b> and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a <b>NULL command</b> to indicate that nothing should be executed.
3	<b>Exception Handling</b> This section starts with the keyword <b>EXCEPTION</b> . This optional section contains <b>exception(s)</b> that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**. Following is the basic structure of a PL/SQL block –

## **DECLARE**

<declarations section>

## **BEGIN**

<executable command(s)>

## **EXCEPTION**

<exception handling>

**END;**

## **PL/SQL Language Elements:**

### **Character Set**

Character set may include the following characters:

- Alphabets, both in upper case [A–Z] and lower case [a–z]
- Numeric digits [0–9]
- Special characters ( ) + - \* / < > = ! ~ ^ ; : . \_ @ % , \_ # \$ & | { } ? [ ]
- Blank spaces, tabs, and carriage returns.

### **Lexical Units**

A line of PL/SQL program contains groups of characters known as lexical units, which can be classified as follows:

- Delimiters
- Identifiers
- Literals
- Comments

### **Delimiters**

The delimiters are used to identify the operators such as +, -, \* etc.

### **Identifiers**

User defined names such as constants, variables, cursors, cursor variables, subprograms, etc. Identifiers can consist of alphabets, numerals, dollar signs, underscores and number signs only. An identifier cannot contain more than 30 characters.

### **Literals**

A literal is an explicitly defined character, string, numeric or Boolean value

#### **1. Numeric Literals**

A numeric literal is an integer or a real value.

An integer literal may be a positive, negative, or unsigned without a decimal point.

Ex: 100 006 -10 +10

A real literal is a positive, negative, or unsigned with a decimal point.

Ex: 0.0 -19.0 3.56219 +43.99

#### **2. Character Literals**

A character literal is an individual character enclosed by single quotes (apostrophes).

Ex: 'A' '@' '5' '?' ',' '('

#### **3. String Literals**

A string literal is enclosed within double quotes.

Ex: "Good Morning!" "04-MAY-00"

#### **4. Boolean Literals**

Boolean literals are the predefined values TRUE, FALSE, and NULL. Keep in mind Boolean literals are values, not strings.

## **Comments**

Comments are used in the PL/SQL program to improve the readability and understandability of a program. Generally, comments are used to describe the purpose and use of each code segment. A PL/SQL comment may be a single-line or multiline.

## Single-Line Comments

Single-line comments begin with a double hyphen (–)

**Example:** – start calculations

## Multiline Comments

Multiline comments begin with a slash-asterisk (/\*) and end with an asterisk slash

**Example:** /\* Hello World! \*/

## **Data types in PL/SQL:**

- Char(): Character datatype is used to store alphabets. The maximum number of characters is 2000.
- Varchar(): It is used to store variable length of alpha-numeric data and maximum storage capacity is 2000.
- Varchar2(): It is used to store variable length of alpha-numeric data of maximum 4000 characters.
- Long(): It is used to enter the alpha-numeric data for description of a record. The maximum size is 2 GB.
- Number(): It is used to store the numeric values upto 38 digits.
- Date(): It is used to store the dates in the format mm/dd/yy.
- Boolean(): it is used to store true or false values for a variable.
- %type(): this datatype is used to provide the datatype of a variable or the datatype of a column in a table.  
Ex: balance number(8,2)  
Minibalance balance%type;
- %rowtype(): It store the entire record from a table.  
Ex: emp\_rec emp%rowtype;

## Operators in PL/SQL:

### **Operators Precedence**

The operations within an expression are done in a particular order depending on their precedence.

### Order of operations

<u>operator</u>	<u>operation</u>
** , NOT	- exponentiation, logical negation
+, –	- identity, negation
*, /	- multiplication, division
+, –,	- addition, subtraction, concatenation
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	- comparison
AND	- conjunction
OR	- disjunction

## Control Structure:

Control structure is an essential part of any programming language. It controls the flow of process. Control structure is broadly divided into three categories:

1. Conditional control,
2. Iterative control, and
3. Sequential control

### **1. Conditional Control**

A conditional control structure tests a condition to find out whether it is true or false and accordingly executes the different blocks of SQL statements. Conditional control is generally performed by IF statement.

There are three forms of IF statement.

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSEIF.

- **IF-THEN**

It is the simplest form of IF condition.

**Syn:** IF condition THEN  
Statements-block 1  
END IF;

**Ex:**

```
IF A > B THEN
big := A;
ENDIF;
```

The Statements-block is executed only if the condition is true. If the condition is FALSE or NULL, the Statements-block is skipped and processing continues from statements following END IF statements.

- **IF-THEN-ELSE**

**Syn:** IF condition THEN  
Statements-block 1  
ELSE  
Statements-block 2  
END IF;

**Ex:**

```
IF A > B THEN
big:= A;
ELSE
big:= B;
ENDIF;
```

First the Statements-block 1 is executed only if the condition is true. If the condition is FALSE or NULL, the Statements-block 2 is executed and processing continues from statements following END IF statements.

The sequence of statements in the ELSE clause is executed only if the condition is FALSE or NULL.

- **IF-THEN-ELSIF**

**Syn:** IF condition1 THEN  
Statements-block 1  
ELSIF condition2 THEN  
Statements-block 2  
ELSE  
Statements-block 3  
END IF;

In the previous constructs of IF, we can check only one condition, whether it is true or false. There is no provision if we want to check some other conditions if first condition evaluates to FALSE; for this purpose third form of IF statement is used.

## 2. **Iterative control:**

In iterative control a group of statements are executed repeatedly till certain condition is true, and control exits from loop to next statement when the condition becomes false. There are mainly three types of loop statements:

- LOOP
- WHILE-LOOP
- FOR-LOOP

- **LOOP:**

- **LOOP**

LOOP is the simplest form of iterative control. It encloses a sequence of statements between the keywords LOOP and END LOOP.

**Syn: LOOP**  
**sequence of statements**  
**END LOOP;**

With each iteration of the loop, the sequence of statements gets executed, then control reaches at the top of the loop. But a control structure like this gets entrapped into infinite loop. To avoid this it is must to use the key word EXIT and EXIT-WHEN.

- **LOOP – EXIT**

An EXIT statement within LOOP forces the loop to terminate unconditionally and passes the control to next statements.

**Syn: LOOP**  
**IF condition1 THEN**  
**Sequence of statements1**  
**EXIT;**  
**ELSIF condition2 THEN**  
**Sequence of statements2**  
**EXIT**  
**ELSE**  
**Sequence of statements3**  
**EXIT;**  
**END IF;**  
**END LOOP;**

- **LOOP – EXIT WHEN**

The EXIT-WHEN statement terminates a loop conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop terminates and control passes to the next statement after the loop.

**Syn: LOOP**  
**EXIT WHEN condition**  
**Sequence of statements**  
**END LOOP**  
**Example**

- **WHILE-LOOP**

The WHILE statement with LOOP checks the condition. If it is true then only the sequence of statements enclosed within the loop gets executed. Then control resumes at the top of the loop and checks the condition again, the process is repeated till the condition is true. The control passes to the next statement outside the loop for FALSE or NULL condition.

**Syn: WHILE condition LOOP**  
**Sequence of statements**  
**END LOOP;**

- **FOR-LOOP**

FOR loops iterate over a specified range of integers. The range is part of iteration scheme, which is enclosed by the keywords FOR and LOOP. A double dot (..) serves as the range operator.

**Syn: FOR counter IN lower limit .. higher limit LOOP**  
**sequence of statements**  
**END LOOP;**

The range is evaluated when the FOR loop is first entered and is never re-evaluated. The sequence of statements is executed once for each integer in the range. After every iteration, the loop counter is incremented.

```

SET SERVEROUTPUT ON

DECLARE
C NUMBER(4);
SUMN NUMBER(6) := 0;

BEGIN

FOR C IN 1 .. 10 LOOP
SUMN := SUMN + C;
END LOOP;

DBMS_OUTPUT.PUT_LINE('SUM UPTO 10 : ' || ' ' || SUMN);

END;

```

### 3. Sequential Control

The sequential control unconditionally passes the control to specified unique label; it can be in the forward direction or in the backward direction. For sequential control GOTO statement is used. Overuse of GOTO statement may increase the complexity, thus as far as possible avoid the use of GOTO statement.

**Syn: GOTO label;**

```

.....
.....
<<label>>
Statement

```

### Procedures and Functions:

#### Procedure:

A procedure is a subprogram that performs some specific task, and stored in the data dictionary. A procedure must have a name, so that it can be invoked or called by any PL/SQL program that appears within an application. Procedures can take parameters from the calling program and perform the specific task.

#### **Benefits of Procedures:**

1. It modifies one routine to affect multiple applications.
2. It modifies one routine to eliminate duplicate testing.
3. It ensures that related actions are performed together.
4. It reduces the number of calls to the database.

#### **Creating Procedures:**

##### **Syntax:**

```

CREATE OR REPLACE PROCEDURE [schema.] package name
[(argument {IN, OUT, IN OUT} data type,.....)] {IS, AS}
[local variable declarations]
BEGIN
executable statements
EXCEPTION
exception handlers
END [procedure name];

```

A procedure consists of two parts

1. Specification
2. Body.

#### **1. Specification:**

The specification starts with keyword PROCEDURE and ends with parameter list or procedure name. The procedures may accept parameters or may not. Procedures that do not accept parameters are written parentheses.

## 2. Body:

The body starts with the keyword IS and ends with keyword END. The procedure body is further subdivided into three parts:

- **Declarative part:** It consists of local declarations placed between keywords IS and BEGIN.
- **Executable part:** It consists of actual logic of the procedure, included between keywords BEGIN and EXCEPTION. At least one executable statement is a must in the executable portion of a procedure. Even a single NULL statement will do the job.
- **Error/Exception handling part:** It is an optional part placed between EXCEPTION and END.

```
SQL> DESC EMP;
Name                               Null?    Type
-----
EMPID                               NOT NULL VARCHAR2(6)
EMPNAME                             VARCHAR2(30)
SAL                                  NUMBER(6)
```

```
SQL> CREATE OR REPLACE PROCEDURE raise_sal
2 (EID IN EMP.EMPID%TYPE)
3 IS
4 BEGIN
5 UPDATE EMP
6 SET SAL = SAL * 1.25 WHERE EMPID=EID;
7 END raise_sal;
8 /
```

Procedure created.

```
SQL> SELECT * FROM EMP;
```

EMPID	EMPNAME	SAL
E101	KARTHIKEYAN	5000
E102	ANAND	2500
E103	RAJA	1900

```
SQL> EXECUTE raise_sal('E101');
```

PL/SQL procedure successfully completed.

```
SQL> SELECT * FROM EMP;
```

EMPID	EMPNAME	SAL
E101	KARTHIKEYAN	6250
E102	ANAND	2500
E103	RAJA	1900

## Removing a Procedure:

To remove a procedure completely from the database, following command is used:

**Syn:** DROP PROCEDURE <PROCEDURE NAME>;

**Ex:** SQL> DROP PROCEDURE raise\_sal;

Procedure dropped.

## Function:

A Function is similar to procedure except that it must return one and only one value to the calling program. Besides this, a function can be used as part of SQL expression, whereas the procedure cannot.

## Difference Between Function and Procedure

1. A procedure never returns a value to the calling portion of code, whereas a function returns exactly one value to the calling program.
2. As functions are capable of returning a value, they can be used as elements of SQL expressions, whereas the procedures cannot. However, user-defined functions cannot be used in CHECK or DEFAULT constraints and cannot manipulate database values, to obey function purity rules.

3. It is mandatory for a function to have at least one RETURN statement, whereas for procedures there is no restriction. A procedure may have a RETURN statement or may not. In case of procedures with RETURN statement, simply the control of execution is transferred back to the portion of code that called the procedure.

```
CREATE OR REPLACE FUNCTION [schema.] functionname
[(argument IN datatype, . . . .)] RETURN datatype {IS,AS}
[local variable declarations];
BEGIN
executable statements;
EXCEPTION
exception handlers;
END [functionname];
```

where RETURN datatype is the datatype of the function's return value. It can be any PL/SQL datatype. The function has two parts

1. Function specification and
2. Function body.

**1. Function specification :** The function specification begins with keyword FUNCTION and ends with RETURN clause which indicates the datatype of the value returned by the function.

**2. Function body :** Function body is enclosed between the keywords IS and END. Sometimes END is followed by function name, but this is optional.

Like procedure, a function body is composed of three parts:

- Declarative Part
- Executable Part
- Error/Exception Handling Part.

- A function can have multiple return statements, but can return only one value.
- In procedures, return statement cannot contain any expression, it simply returns control back to the calling code.
- However in functions, return statement must contain an expression, which is evaluated and sent to the calling code.

```
SQL> DESC EMP;
Name                               Null?    Type
-----
EMPID                               NOT NULL VARCHAR2(6)
EMPNAME                             VARCHAR2(30)
SAL                                  NUMBER(6)

SQL> CREATE OR REPLACE FUNCTION get_sal
2 (EID IN EMP.EMPID%TYPE)
3 RETURN NUMBER
4 IS
5 EMPSAL EMP.SAL%TYPE :=0;
6 BEGIN
7 SELECT SAL INTO EMPSAL
8 FROM EMP WHERE EMPID=EID;
9 RETURN(EMPSAL);
10 END;
11 /

Function created.

SQL>
```

```
SQL> SELECT * FROM EMP;

EMPID  EMPNAME          SAL
-----
E101   KARTHIKEYAN     6250
E102   ANAND            2500
E103   RAJA             1900

SQL> SELECT GET_SAL('E102') FROM DUAL;

GET_SAL('E102')
-----
                2500

SQL> |
```

### Calling a Function

Syn: Select function\_name(argument) from dual;

Ex: select get\_sal('E102') from dual;



## **Removing a Function**

To remove a function, use following command:

Syn: DROP FUNCTION <FUNCTION NAME>;

Ex: Drop function get\_sal;

## **Packages**

A package is a group of related procedures and functions stored together and sharing common variables, as well as local procedures and functions.

## **Advantages of Packages**

Packages offer a lot of advantages. They are as follows.

1. Stored packages allow us to sum up (group logically) related stored procedures, variables, and data types
2. Grouping of related procedures, functions, etc. in a package also make privilege management easier.
3. Package helps in achieving data abstraction. Package body hides the details of the package contents and the definition of private program objects so that only the package contents are affected if the package body changes.
4. Packages provide better performance than stored procedures and functions because public package variables persist in memory for the duration of a session. So that they can be accessed by all procedures and functions that try to access them.
5. Packages allow overloading of its member modules. More than one function in a package can be of same name. The functions are differentiated, depending upon the type and number of parameters it takes

## **Units of Packages**

As described earlier, a package is used to store together, the logically related PL/SQL units. In general, following units constitute a package.

- Procedures
- Functions
- Triggers
- Cursors
- Variables

## **Parts of Package**

A Package has two parts. They are:

- Package specification
- Package body

## **Package Specification**

The specification declares the types, variables, constants, exceptions, cursors, and subprograms that are public and thus available for use outside the package. In case in the package specification declaration there is only types, constants, exception, or variables, then there is no need for the package body because package specification are sufficient for them. Package body is required when there is subprograms like cursors, functions, etc.

## **Package Body**

The package body fully defines subprograms such as cursors, functions, and procedures. All the private declarations of the package are included in the package body. It implements the package specification.

## **Creating a Package**

A package consists of package specification and package body. Hence creation of a package involves creation of the package specification and then creation of the package body.

The package specification is declared using the CREATE PACKAGE command.

The syntax for package specification declaration is as follows.

```
CREATE[OR REPLACE] PACKAGE <package name>  
[AS/IS]  
PL/SQL package specification
```

All the procedures, sub programs, cursors declared in the CREATE PACKAGE command are described and implemented fully in the package body along with private members. The syntax for declaring a package body is as follows:

```
CREATE[OR REPLACE] PACKAGE BODY <package name>  
[AS/IS]  
PL/SQL package body
```

Member functions and procedures can be declared in a package and can be made public or private member using the keywords public and private. Use of all the private members of the package is restricted within the package while the public members of the package can be accessed and used outside the package.

### Referencing Package Subprograms

- Once the package body is created with all members as public, we can access them from outside the program.
- To access these members outside the packages we have to use the dot operator, by prefixing the package object with the package name.

Syn: <PACKAGE NAME>.<VARIABLE NAME>

To reference procedures we have to use the syntax as follows:

```
EXECUTE <package name>.<procedure name(variables)>;
```

### Removing a Package

A package can be dropped from the database just like any other table or database object. The exact syntax of the command to be used for dropping a package is:

```
DROP PACKAGE <PACKAGE NAME>;
```

### Cursor:

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

There are two types of cursors –

- Implicit cursors
- Explicit cursors

### Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

The implicit cursor has 4 attributes such as

1. **%FOUND**
2. **%ISOPEN**
3. **%NOTFOUND**
4. **%ROWCOUNT**

#### 1. **%FOUND**

Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

#### 2. **%NOTFOUND**

The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.

#### 3. **%ISOPEN**

Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.

#### 4. **%ROWCOUNT**

Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

## **Explicit Cursors**

Explicit cursors are user defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block

Syn: `CURSOR cursor_name IS select_statement;`

Explicit cursor includes the following steps

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

### **Declaring the Cursor**

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example –

```
CURSOR c_customers IS
SELECT id, name, address FROM customers;
```

### **Opening the Cursor**

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows –

```
OPEN c_customers;
```

## Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows –

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

## Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows –

```
CLOSE c_customers;
```

## Q. What is Trigger?

A trigger is a PL/SQL block structure which is fired when a DML statements like **Insert, Delete, Update** is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

## Syntax for Creating a Trigger

```
CREATE [OR REPLACE ] TRIGGER trigger_name  
{ BEFORE | AFTER | INSTEAD OF }  
{ INSERT [OR] | UPDATE [OR] | DELETE }  
[OF col_name]  
ON table_name  
[REFERENCING OLD AS o NEW AS n]  
[FOR EACH ROW]  
WHEN (condition)  
BEGIN  
--- sql statements  
END;
```

---

## Type of Triggers

- BEFORE Trigger :** It executes before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.
- AFTER Trigger :** It executes after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is execute as soon as followed by the code of trigger before performing Database operation.
- ROW Trigger :** It fires for each and every record which are performing INSERT, UPDATE, DELETE from the database table. If row deleting is define as trigger event, when trigger file, deletes the five rows each times from the table.
- Statement Trigger :** It fires only once for each statement. If row deleting is define as trigger event, when trigger file, deletes the five rows at once from the table.
- Combination Trigger :** These are combination of two trigger types,

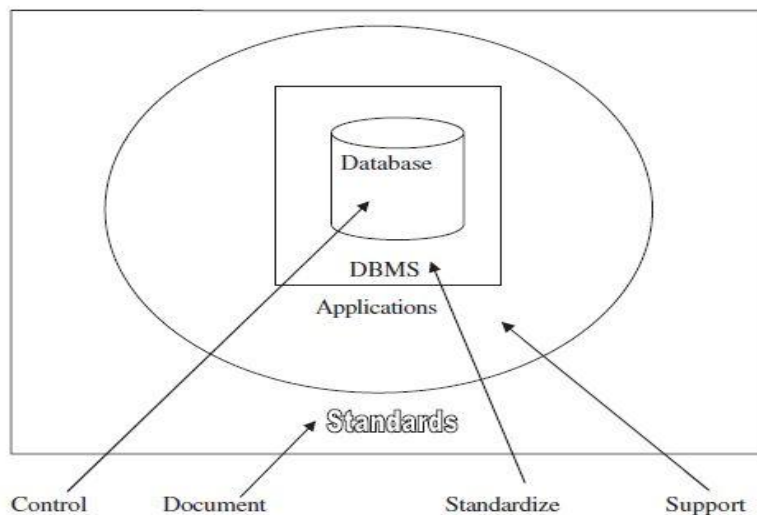
- **Before Statement Trigger :** Trigger fire only once for each statement before the triggering DML statement.
- **Before Row Trigger :** Trigger fire for each and every record before the triggering DML statement.
- **After Statement Trigger :** Trigger fire only once for each statement after the triggering DML statement executing.
- **After Row Trigger :** Trigger fire for each and every record after the triggering DML statement executing.

## Q. What are the responsibilities(role) of Database Administrator(DBA)?

### Database Administrator (DBA)

Database Administrator is a person having central control over data and programs accessing that data. The database administrator is a manager whose responsibilities are focused on management of technical aspects of the database system. The objectives of database administrator are given as follows:

1. To control the database environment
2. To standardize the use of database and associated software
3. To support the development and maintenance of database application projects
4. To ensure all documentation related to standards and implementation is up-to-date



### **Responsibilities of Database Administrator (DBA)**

- Installing and updating the application tools
- Allocating system storage and planning further storage requirements for the database system
- Creating primary database storage structures after application developers have design an application
- Creating primary object(table, views, index) once application developers have designed an application
- Modifying the database structure as necessary from information given by application developers
- Enroll user and maintaining user access to the database
- Planning for backup and recovery of database information
- Controlling and maintaining user access to the database
- Monitor and optimize the performance of the database
- Maintain archive data on tapes
- Backup and restore the database
- Contact DBMS vendors for technical problems
- Develop a plan for the organization's database/data resource
- Organize the staff and database administration (DBA) functions
- Supervise the DBA function
- Give training to the database Users
- Act as a bridge in communicating between managers, users and application developers.

**Department of Computer Science**  
**III B.Sc – V Semester (V-A)**  
**DBMS Important Questions**

**Essay Questions:**

1. Define DBMS? Explain the advantages of DBMS?
2. Explain about Database Architecture?
3. Explain about Entity Relationship(ER) model?
4. Explain about Extended Entity Relationship Model(EERM)?
5. What are CODD's Rules?
6. Explain about Relational algebra Operators?
7. Explain DDL, DML, DCL & TCL commands with syntax and example?
8. Explain about Set Operators and Joins with syntax and Example?
9. Explain the Control Structures in PL/SQL?
10. Explain the types of Cursors & how to create Cursors?

**Short Questions:**

1. Explain about the evolution of DBMS?
2. Explain the objectives of DBMS?
3. What are the components of DBMS?
4. What are the building blocks of ER diagram? Or What are the building blocks of Data Model?
5. Explain about Specialization and Generalization?
6. What are Advantages & Limitations of Relational algebra?
7. Explain Aggregate functions & Constraints with syntax and example?
8. Explain about views?
9. Explain the structure of PL/SQL?
10. What is Trigger? Explain different types of Triggers?
11. What are the responsibilities(role) of Database Administrator(DBA)?
12. Explain the levels of Data Abstraction? Or Explain the ANSI/SPARK Data model?