# SRI GCSR COLLEGE

**DATABASE MANAGEMENT SYSTEM**


**DEPARTMENT OF COMPUTER SCIENCE**

**SRI GCSR COLLEGE**

**GMR NAGAR**

**RAJAM – 532127**

**UNIT I**

**Overview of Database Management System: Introduction to data, information, database,database management systems, file-based system, Drawbacks of file-Based System, databaseapproach, Classification of Database Management Systems, advantages of database approach,Various Data Models, Components of Database Management System, three schemaarchitecture of data base, costs and risks of database approach.**

**UNIT II**

**Entity-Relationship Model: Introduction, the building blocks of an entity relationshipdiagram, classification of entity sets, attribute classification, relationship degree, relationshipclassification, reducing ER diagram to tables, enhanced entity-relationship model (EERmodel), generalization and specialization, IS A relationship and attribute inheritance,multiple inheritance, constraints on specialization and generalization, advantages of ERmodelling.**

**UNIT III**

**Relational Model: Introduction, CODD Rules, relational data model, concept of key,relational integrity, relational algebra, relational algebra operations, advantages of relationalalgebra, limitations of relational algebra, relational calculus, tuple relational calculus, domainrelational Calculus (DRC), Functional dependencies and normal forms upto 3rd normal form.**

**UNIT IV**

**Structured Query Language: Introduction, History of SQL Standard, Commands in SQL,Data Types in SQL, Data Definition Language, Selection Operation, Projection Operation,Aggregate functions, Data Manipulation Language, Table Modification Commands, JoinOperation, Set Operations, View, Sub Query.**

**UNIT V**

**PL/SQL: Introduction, Shortcomings of SQL, Structure of PL/SQL, PL/SQL LanguageElements, Data Types, Operators Precedence, Control Structure, Steps to Create a PL/SQL, Program, Iterative Control, Procedure, Function, Database Triggers, Types of Triggers**

**Q: Explain the concepts of DBMS.**

A. **Data:** Data is the collection of raw facts and figures.

B. **Information:** The meaningful form of data or processed data is called as information.

C. **Field:** A field is the lowest level of data item of an entity which is alternatively called as attribute of that entity.A field is a single item of data within a database or software program.

For ex, a field may be a customer name, address, or phone number.

D. **Record:** Record is the collection of related fields or data.

| **Ex:** | **S.no** | **Sname** | **Marks** |
|---|---|---|---|
| | 1001 | Raj | 950 |

E. **Database File:** File is a collection of records having the same set of fields arranged in the same sequence.

**Ex:** Student Profile, Student marks, etc.

F. **Database:** Database is a collection of inter-related data which helps in efficient retrieval, insertion, and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc.

For Example, university database organizes the data about students, faculty, and admin staff etc.

G. **DBMS:** The software which is used to manage database is called Database Management System (DBMS). It acts as the interface between users and the database itself. It is a record keeping system. It allows the data to be stored, maintained, manipulated, and retrieved.

For Example, MySQL, Oracle etc. are popular commercial DBMS used in different applications.

It consists of DBMS utilities, database, and data dictionary. Its responsibilities are:

- Multiuser Access Control
- Security Management
- Backup and Recovery Management

**Q: Explain Data hierarch achy?**

**Bit:**

A bit is the smallest unit of measurement used to quantify computer data. It contains a single binary value of 0or 1

**Byte:** A byte is a data measurement unit that contains eight bits, or a series of eight zeros and ones. A single byte can be used to represent 256 different values.

**Field:**In DBMS data is used to store in the form of tables.it containsrecords which is collection of fields. The table columns define what fields are available in each row.

**Record:** Collection of fields in a row is called Record

**File:** Collection of tables forms a file.

**Database:**A database is a collection of files which contains related data arranged in the form of tables.

**Q: What is File-based system? Explain the Drawbacks of file systems?**

A File system is used to store, manage, and retrieve data. A file system can be either manual or computerized. File system allows storing data of different departments in different data files.

A. **Manual File System:** In this system data can be stored and maintained in books, ledgers, and journals etc.
B. **Computerized File System:** In this system data can be stored and maintained in computers. Data can be entered and modified with high speed. Data can be shared partially.

**In File System-**

- Files are stored in different places.
- There is no relation between the files.
- Files are developed by different persons on different locations.

**Drawbacks of file systems:**

A. Uncontrolled data redundancy
B. Inconsistency of data
C. Difficulty in Accessing Data
D. Limited data sharing
E. Poor enforcement of standards

F. Concurrent access

G. Low programmer productivity

H. Security Problems

I. No Backup & Recovery facilities

A. **Uncontrolled redundancy of data:** Redundancy means repeating data. It is possible that the same information may be duplicated in different files. This leads to data redundancy results in memory wastage.

B. **Inconsistency of data:** Because of data redundancythere is a possibility of entering same data differently in different sub-systems. This leads to data inconsistency.

C. **Difficulty in Accessing Data:** Accessing data is not convenient and efficient in file processing system.

D. **Limited Data Sharing:** Data are scattered in various filesalso different files may have different formats and these files may be stored in different folders may be of different departments. So, due to this data isolation, it is difficult to share data among different applications.

E. **Poor enforcement of standards:** Different applications are developed by different persons; each person will follow its own standards of defining field name, field width, and field type. This will create a serious difficulty while modifying programs, sometimes there will be serious errors due to mismatch of fields.

F. **Concurrent Access:**Multiple users are allowed to access data simultaneously. This is for the sake of better performance and faster response. But File system does not support multiple users to access the data at a time.

G. **Low programmer productivity:**-Programmer productivity is a measure of time taken to develop an application. It is inversely proportional to developing time. Because ofFile system like data redundancy, inflexibility, and poor enforcement standards the programmer productivity will become lower.

H. **Security Problems:** File System has no password protection to the data. So unauthorized persons can access the data. So it has limited security.

I. **No Backup & Recovery Facilities:-** Backup means the original data will be stored in different device. If the original data was removed then the backup data will be stored in original place is called recovery.

File System has no backup and recovery facilities.

## Q: Various Objectives of Database Management System?

**A. Mass Storage:**

DBMS can store a lot of data in it. It can store thousands of records in it and one can fetch all that data whenever it is needed.

**B. Removes Duplicity:**

If you have lots of data then data duplicity will occur for sure at any instance. DBMS guarantee it that there will be no data duplicity among all the records. While storing new records, DBMS makes sure that same data was not inserted before.

**C. Multiple Users Access:**

No one handles the whole database alone. There are lots of users who are able to access database. So this situation may happen that two or more users are accessing database. They can change whatever they want, at that time DBMS makes it sure that they can work concurrently.

**D. Data Protection:**

DBMS gives a master level security to their data. No one can alter or modify the information without the privilege of using that data.Information such as bank details, employee's salary details and sale purchase details should always be kept secured. Also all the companies need their data secured from unauthorized use.

**E. Data Backup and recovery:**

Sometimes database failure occurs so there is no option like one can say that all the data has been lost. There should be a backup of database so that on database failure it can be recovered. DBMS has the ability to backup and recover all the data in database.

**F. Everyone can work on DBMS:**

There is no need to be a master of programming language if you want to work on DBMS. Any accountant who is having less technical knowledge can work on DBMS.

**G. Integrity:**

Integrity means your data is authentic and consistent. DBMS has various validity checks that make your data completely accurate and consistence.

**H. Platform Independent**

One can run dbms at any platform. No particular platform is required to work on database management system.

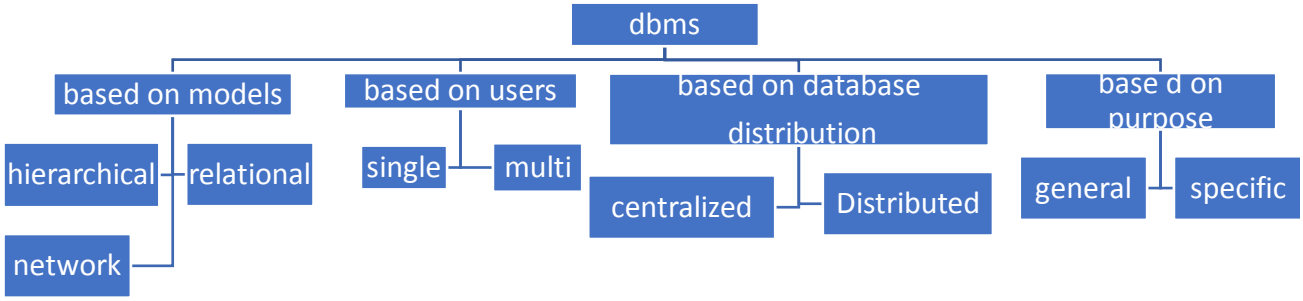
## Q: Evaluations of DBMS?

The following are some of the major evaluations of DBMS

- 1960 – First DBMS designed by Charles Bachman at general electronics known as integrated data source (IDS).
- 1960 – IBM developed the information management system called IBM.
- 1970 – Edgacodd from IBM create a relational data model
- 1976 – Peterchan presented the Entity relationship model
- 1980 – SQL developed by IBM became the standard query language for databases becomes a widely accepted database component
- 1985- Object-oriented DBMS develops.
- 1990s- Incorporation of object-orientation in relational DBMS.
- 1991- Microsoft ships MS access, a personal DBMS and that displaces all other personal DBMS products.
- 1995: First Internet database applications
- 1997: XML applied to database processing. Many vendors begin to integrate XML into DBMS products.

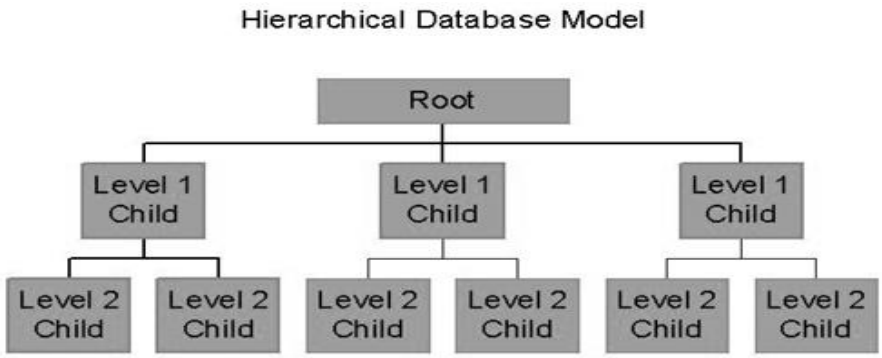
## Q: Explain about Classification of Database Management System?

Database management systems can be classified based on several criteria such as the based on data model, no of users , database distribution and based on purpose.

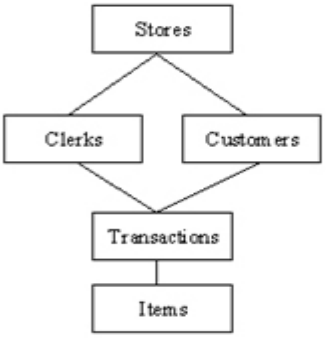



**Based on Model:** Depending on the model they are using classified into Hierarchical, Relational, Network models

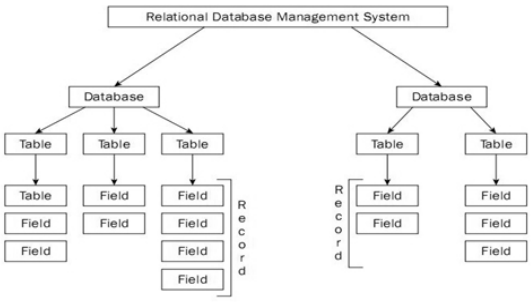
**Hierarchical database** – In this model, the information stored in the form of groups of parent or child relationships , which is similar to the structure of a tree.




**Network Model:**The data record are linked to one another with the help of pointers. Childs can contain multiple owner relation ship



**Relational Model**: RDBMS organizes the data in the form of tables which contains relations among the tables.




**Based on Users:**

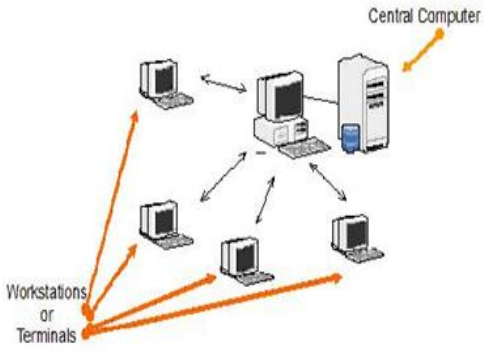
**Single user** – As the name itself indicates it can support only one user at a time. It is mostly used with the personal computer on which the data resides accessible to a single person. The user may design, maintain and write the database programs.

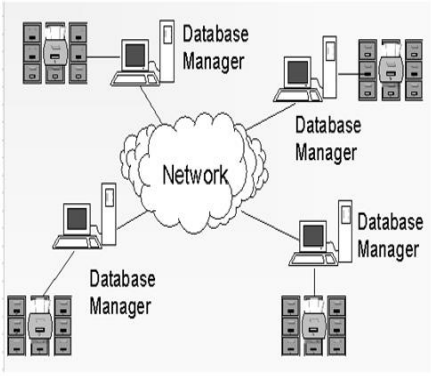
**Multiple users** – It supports multiple users concurrently simultaneously. Data can be both integrated and shared.

**Based on Database Distribution:** based on the classification it is divided into centralized ,distributed

**Centralized database system** – The DBMS and database are stored at the single site that is used by several other systems too. We can simply say that data here is maintained on the centralized server.

**Distributed database system** – In this data and the DBMS software are distributed over several sites but connected to the single computer.



**Based on purpose:** they are classified into general purpose or specific purpose

**General Purpose:**the DBMS which is designed for reservations,airlinesetc are called general purpose.

**Spicificpurpose:**the DBMs which id designed for a specific purpose marketing,bankingetc

**Based on the cost**

**Low cost DBMS** : The cost of these systems vary from $100 to $3000.

**Medium cost DBMS** :Cost varies from $10000 to $100000.

**High cost DBMS** : Cost pf these systems are usually more than $100000..

**Based on the access**

This classification simply based on the access to data in the database systems.

**Sequential access** – One after the other.

**Direct access**- all at once

**Q: What is data model? Explain various types of database models?**

Model is an abstraction of reality. The purpose of the data model is to convey the details of the system for better understanding of the organization of data. There are five types of data models. They are:

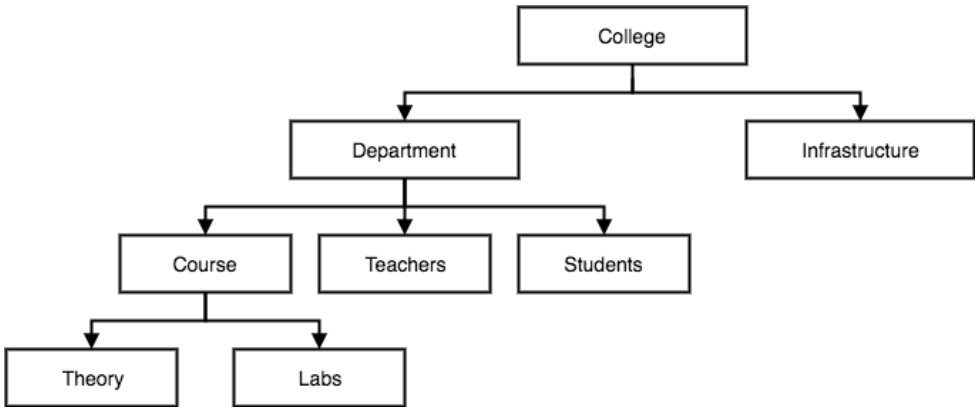
A. Hierarchical data model
B. Network data model
C. Relational Data model
D. Object oriented data model
E. ER model
F. EER model

**Hierarchical Model**

- This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked.
- This model was developed in the end of 1960 to manage large amounts of data for Complex projects.

- The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.
- In this model, a child node will only have a single parent node.
- This model efficiently describes many real-world relationships like index of a book, recipes etc.
- In hierarchical model, data is organized into tree-like structure with one one-to-many relationship between two different types of data,

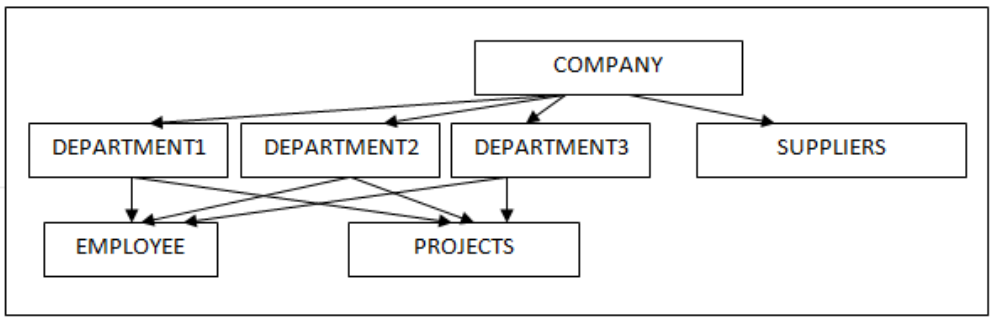
**for example,** one department can have many courses, many professors and of-course many students.




**A. Network Data Model:**

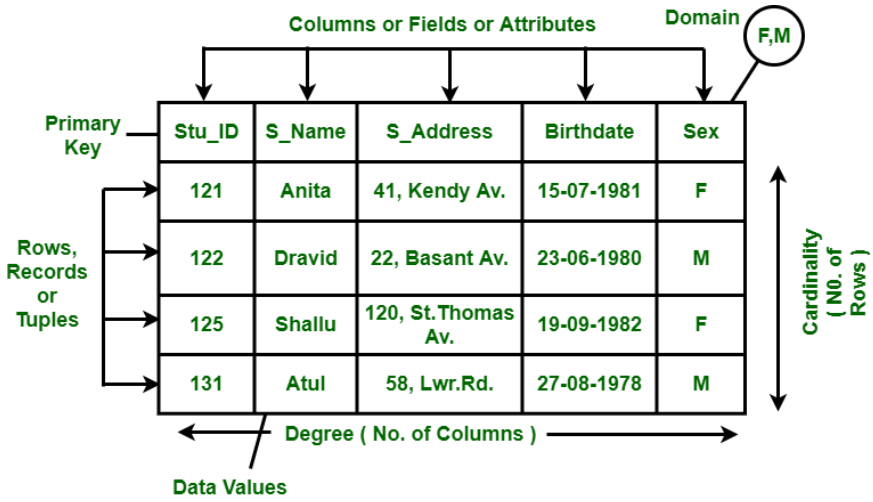
- The network model was implemented in the early 1978.
- This is an extension of the Hierarchical model.
- In this model data is organized more like a graph, and are allowed to have more than one parent node.

**Ex:** An employ work for two departments is not possible in hierarchal model, but here it is possible



## B. Relational Data Model

- The relational data model was developed by E.F. Codd in 1970..
- Itrepresents the database as a collection of relations.
- A relation is a table of values. Every row in the table represents a collection of related data values.
- It organizes records in form of table and relationships between tables are using common fields.




It is the most popular data model of the present day because of the following reasons:

- It is simple to implement.
- It has simple terminology.
- It uses the simple concept of primary key and secondary key to connect any two files.
- In reality almost all the databases are developed based on relational data model. Some of them are Dbase, FoxPro, Ms-Access and Oracle etc.

## C. Object oriented data model:

Object oriented data model is based upon real world situations. These situations are represented as objects, with different attributes. These entire objects have multiple relationships between them.

Elements of Object oriented data model are:

**Objects:** The real world entities and situations are represented as objects

**Attributes and Method:** Every object has certain characteristics. These are represented using Attributes. The behavior of the objects is represented using Methods.

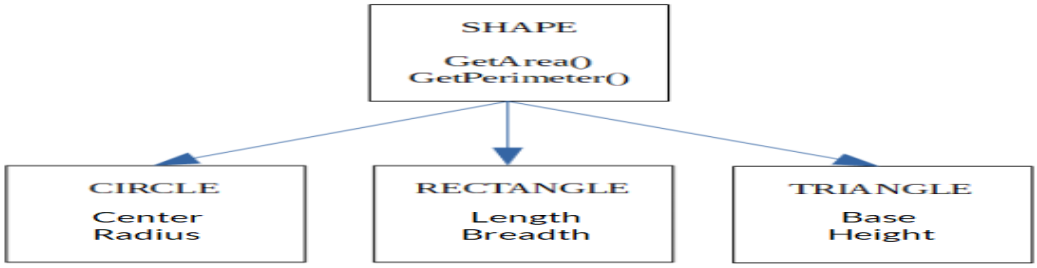
**Class:** Similar attributes and methods are grouped together using a class. An object can be called as an instance of the class.

**Inheritance:** A new class can be derived from the original class. The derived class contains attributes and methods of the original class as well as its own.

**Example In this Object Oriented data model-**

- Shape is a class. Circle, Rectangle and Triangle are all objects in this model.
- Circle has the attributes Center and Radius.
- Rectangle has the attributes Length and Breath
- Triangle has the attributes Base and Height.

The objects Circle, Rectangle and Triangle inherit from the class Shape.



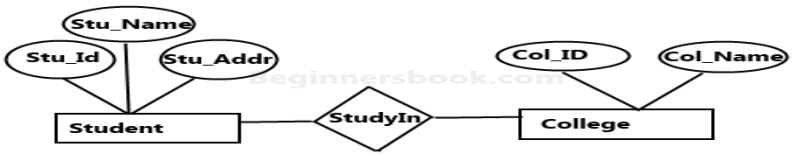

**D. Entity Relationship Data Model (ER Data Model):**

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes.

In this model

- **Rectangle** : Represents Entity sets.
- **Ellipses** : Attributes
- **Diamonds** : Relationship Set
- **Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set




**E. Extended Entity-Relationship (EE-R) Model:**

EER is a high-level data model that incorporates the extensions to the original ER model that represent the requirements and complexities of complex database.

In addition to ER model concepts EE-R includes –

- Subclasses and Super classes.
- Specialization and Generalization.
- Category or union type.
- Aggregation.

**Q: Explain the Advantages of Database Management System?**

A Database Management System (DBMS) is defined as the software system that allows users to create, read, update and delete data in database. It is a layer between programs and data.

Compared to the File Based Data Management System, Database Management System has many advantages. Some of these advantages are given below.

**A. Reducing Data Redundancy**

The file based data management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.

This is prevented in a database as there is a single database and any change in it is reflected immediately.

**B. Consistency of data:-**

In Database the repeated data will be minimized then it always gives consistent results.

**C. Sharing of Data:**

In a database, the users of the database can share the data among themselves. There are various levels of authorisation to access the data, and consequently the data can only be shared based on the correct authorisation.

Many remote users can also access the database simultaneously and share the data between themselves.

**D. Concurrent Access:-**

Database supports multiple users to access the data. Multiple users can access the data at a time.

**E. Data Security**

Data Security is vital concept in a database. Only authorised users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorised users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

**F. Backup and Recovery**

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. Moreover, it also restores the database after a crash or system failure.

**G. Better enforcement of standards:-**

Database is a collection of related files. These files are developed by different persons under the guidance of Data Base Administrator. They will give the same the field name, field type and field size in the files.

**Increased programmer productivity:-**

Programmer productivity is a measure of time taken to develop an application. Database has so many advantages like controlled data redundancy, consistent of data etc. So programming productivity is always high

Q: Explain the differences between DBMS and File System?

## Difference Between File System and DBMS

| DBMS | File System |
|---|---|
| DBMS is a collection of data. In DBMS, the user is not required to write the procedures. | File system is a collection of data. In this system, the user has to write the procedures for managing the database. |
| DBMS gives an abstract view of data that hides the details. | File system provides the detail of the data representation and storage of data. |
| DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure. | File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost. |
| DBMS provides a good protection mechanism. | It is very difficult to protect a file under the file system. |
| DBMS contains a wide variety of sophisticated techniques to store and retrieve the data. | File system can't efficiently store and retrieve the data. |
| DBMS takes care of Concurrent access of data using some form of locking. | In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information. |
| Redundancy is control in DBMS | Redundancy not control in file system |
| Unauthorized access is restricted in DBMS | Not in the file system |
| DBMS provide back up and recovery | Data lost in file system can't be recovered. |
| DBMS provide multiple user interfaces | Data is isolated in file system. |
| A database management system coordinates both the physical and the logical access to the data | file-processing system coordinates only the physical access. |
| A database management system is designed to allow flexible access to data (i.e. queries) | A file-processing system is designed to allow predetermined access to data (i.e. compiled programs). |
| A database management system is designed to coordinate multiple users accessing the same data at the same time. | A file-processing system is usually designed to allow one or more programs to access different data files at the same time. |

## Q: Explain about ANSI-SPARC Architecture

In 1975, ANSI-SPARC realized the need for a three-level approach with the three levels of abstraction comprises of an external, a conceptual, and an internal level.

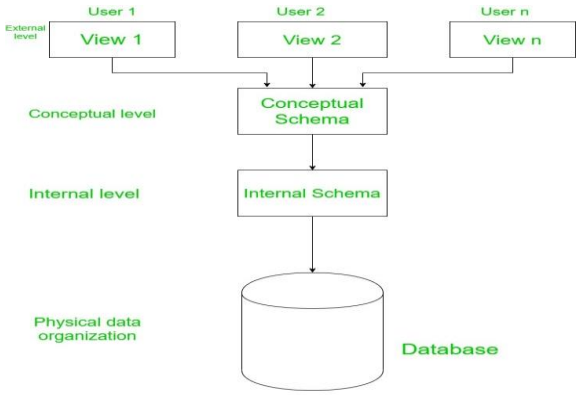
The three-level architecture aims to separate each user's view of the database from the way the database is physically represented.

**Internal level:**

- This is the lowest level of data abstraction. It is also known as physical level.
- It describes how the data are actually stored on storage devices.
- It provides internal view of physical storage of data.
- It deals with Data Compression and Encryption techniques if used.

**Conceptual level:**

- This is the next higher level than internal level of data abstraction.
- It describes what data are stored in the database and what relationships exist among those data.
- It is also known as logical level.
- It hides low level complexities of physical storage.
- Database administrator, application developers and designers work at this level to determine what data to keep in database.




The ANSI-SPARC three-level architecture

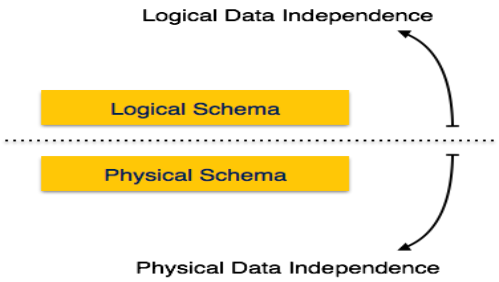
**External Level:**

- This is the highest level of data abstraction. It is also known as a view level.
- It describes only part of the entire database that a end user concern.
- End users need to access only part of the database rather than entire database.
- Different user needs different views of database and so, there can be many view level abstractions of the same database.

**Q: Explain about Data Independence concept ?**

**Data Independence**

A database system normally contains a lot of data and it needs to change to satisfy the requirements of the users. If the entire data is dependent, it becomes difficult job.With the help of data independence when we change data in lower levels the upper levels are un effected.




Data Independence are classified into 2 types

1.logical data independence

2.Physical data independence

**Logical Data Independence**

Logical data is data about database, that is, it stores information about how data is managed inside. If we do some changes on table format, it should not change the data residing on the disk is called logical independence.

**Physical Data Independence**

All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the logical schema.

**Q: Explain the Components and interface of DBMS?**

DBMS acts as interface between the user and database. The user requests the DBMS to perform the various operations on the database. The components will perform these requests and provide necessary information to the user. Database Management contains five major components

a. Hardware
b. Software
c. Data
d. Users

**1.Hardware:-**

- The hardware can range from a single personal computer to a network of computers

- The particular hardware depends on the requirement of the organisation and the DBMS used.
- The DBMS run only on the particular operating system.
- It requires a minimum amount of main memory and disk space to run, but this minimum configuration may not necessarily give acceptable performance.

**2.Software:-**

- This is the set of programs used to control and manage the overall database.
- This includes the DBMS software itself, the operating system, the networking software used to share the data among the users and the application program used to access that a in the DBMS..

**3. Data:-**

DBMS exists to collect, store, process and access the data. It is the most important component. The main features of data in the database are

- The data in the database is well organised.
- The data in the database is related.
- The data is accessible in different orders without great difficulty.

**Data dictionary:**

Data Dictionary consists of database metadata i.e, Data about Data.

**Data Dictionary consists of the following information −**

- Name of the tables in the database
- Constraints of a table i.e. keys, relationships, etc.
- Columns of the tables that related to each other which specifies field name, field type and field size.

**4. Users:-**

The users are the people who perform the different operations on the database in the database system.

There are different kinds of people who play different roles in the database system

**a) Database Administrator (DBA) :**

Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of database.The DBA will then create a new account id and password for the user if he/she need to access the data base.

**b)Naive / Parametric End Users :**

Parametric End Users are the unsophisticated who don't have any DBMS knowledge but they frequently use the data base applications in their daily life to get the desired results.

**c)System Analyst :**

System Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.

**d)Sophisticated Users :**

Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database. They can develop their own data base applications according to their requirement.

**e)Data Base Designers :**

Data Base Designers are the users who design the structure of data base which includes tables, indexes, views, constraints, triggers, stored procedures. He/she controls what data must be stored and how the data items to be related.

**f)Application Program :**

Application Program are the back end programmers who writes the code for the application programs.They are the computer professionals. These programs could be written in Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc.

**g)Casual Users / Temporary Users :**

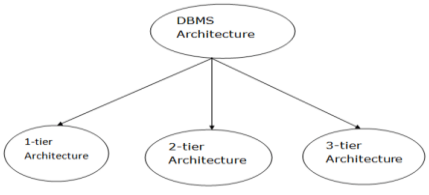
Casual Users are the users who occasionally use/access the data base but each time when they access the data base they require the new information

**Q:Explain DBMS Architecture?**

The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.

The client/server architecture consists of many PCs and a workstation which are connected via the network.DBMS architecture depends upon how users are connected to the database to get their request done.

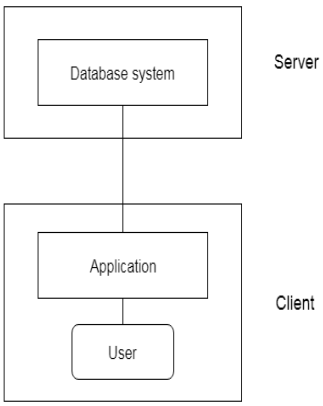
**Types of DBMS Architecture**



**1-Tier Architecture:**

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

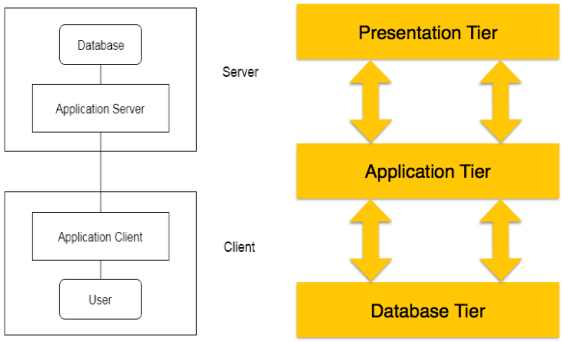
**2-Tier Architecture:**




- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: ODBC, JDBC are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

**3-Tier Architecture:**

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.



A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

**Database (Data) 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.

**Application (Middle) Tier** – At this tier reside the application server and the programs that access the database., the application layer sits in the middle and acts as a mediator between the end-user and the database.

**User (Presentation) Tier** – End-users operate on this tier and they dont know nothing about any existence of the database beyond this layer. All views are generated by applications that reside in the application tier.

**Q:Functions and responsibilities of DBAs?**

**DBA:** person in the organization who controls the design and the use of the database refers as DBA.

**Schema Definition:**

- The DBA definition the logical Schema of the database.A Schema refers to the overall logical structure of the database.
- According to this schema, database will be developed to store required data for an organization.

**Storage Structure and Access Method Definition:**

- The DBA decides how the data is to be represented in the stored database.

**Assisting Application Programmers:**

- The DBA provides assistance to application programmers to develop application programs.

**Physical Organization Modification:**

- The DBA modifies the physical organization of the database to reflext the changing needs of the organization or to improve performance.

**Approving Data Access:**

- The DBA determines which user needs access to which part of the database.
- According to this,various types of authorizations are granted to different users.

**Monitoring Performance:**

- The DBA monitors performance of the system.The DBA ensures that better performance is maintained by making changes in physical or logical schema if required.

**Backup and Recovery:**

- Database should not be lost or damaged.
- The DBA ensures this periodically backing up the database on magnetic tapes or remote servers.
- In case of failure, such as virus attack database is recovered from this backup.

**Q:Explain about Data dictionary?**

- Dictionary is also known as a system catalog.
- It is a centralized store of information about the database.
- Which contains information about the tables, Fields(attributes), data types, primary keys, indexes, the joints which have been established between these tables, referential integrity, cascade update, cascade delete..etc
- The information stored in the data dictionary is called the meta data

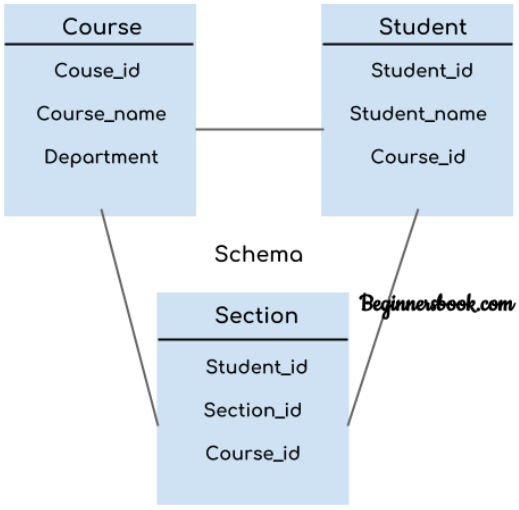
**Metadata:-**

The information about the data in a database is called the metadata

**Q: Explain about DBMS Schema?**

**Definition of schema**: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

**For example:** In the following diagram, we have a schema that shows the relationship between three tables: Course, Student and Section. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view(design) of a database as shown in the diagram below.




**physical schema:**The design of a database at physical level is called **physical schema**, how the data stored in blocks of storage is described at this level.

**logical schema:** Design of database at logical level is called **logical schema**, programmers and database administrators work at this level, at this

level the internal details such as implementation of data structure is hidden at this level (available at physical level).

**view schema.**: Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

**Definition of instance**: The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

**Q: Explain about DBMS Vendors and Their Products?**

There are many different vendors that currently produce relational database management systems (RDBMS).

The leading vendors of RDBMS are listed below:

| **** RDBMS Vendors **** | **** RDBMS Product **** |
|---|---|
| Computer Associates | INGRES |
| IBM | DB2 |
| INFORMIX Software | INFORMIX |
| Oracle Corporation | Oracle |
| Microsoft Corporation | MS Access |
| Microsoft Corporation | SQL Server |
| MySQL AB | MySQL |
| NCR Teradata | |
| PostgreSQL Dvlp Grp | PostgreSQL |
| Sybase | Sybase 11 |

UNIT-II

## Entity–relationship model:

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). It is a high-level data model. An ER model is a design or blueprint of a database that can later be implemented as a database.

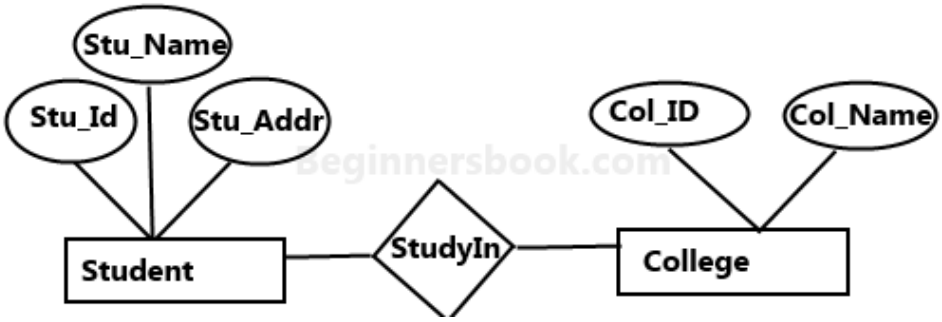
The main components of E-R model are Entity set, Attributes and Relationships.

## ER Diagram:

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes.

In terms of DBMS, an entity is a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

In the following diagram we have two entities Student and College and their relationship. Student entity has attributes such as Stu_Id,Stu_Name&Stu_Addr and College entity has attributes such as Col_ID&Col_Name.




## Q: Explain Building blocks of Entity Relationship Diagram?

The components of the entity-relationship model are the building block which helps in the generation of an ER diagram, which finally results in the design of the logical structure of a database. There are three basic components of the Entity-Relationship Model.

All these components have definite diagrammatical representations that are used for the generation of ER Diagram.

**A. Entity:**

An entity can be called as the basic object which represents the ER Model. Entities are Real world objects or things or articles or pieces that have an existence without any dependence on any other object.

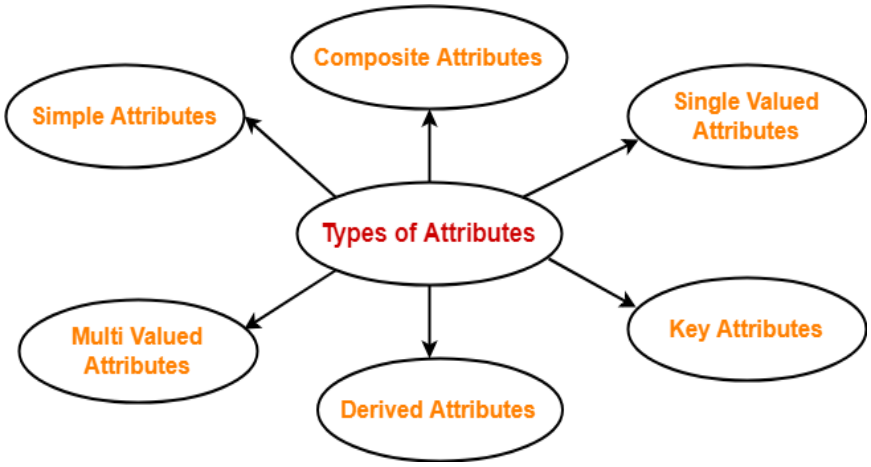
**For example:** Bank account, Student, employ all these are entities have their own properties which are known as attributes.

**B. Attribute:**

Attributes describe the properties and characteristics of the entity.

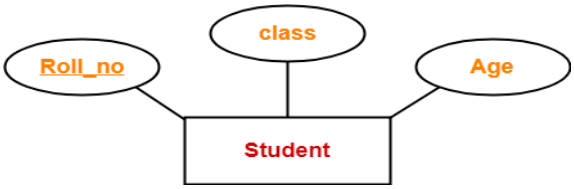
**For example:** Attributes of Bank account can be " Account No, Account Holder Name, Balance" etc.

There exist seven types of attributes for specific types of entities.



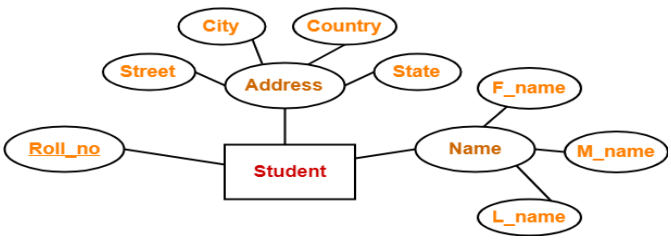

**a) Simple Attribute:** Simple attribute are those attributes which cannot be divided further because of their atomic nature.

For Example: The roll number of a student.



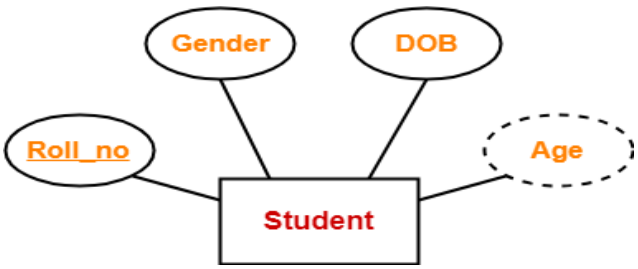

**b) Composite attribute:** An attribute which can be divided into components is a composite attribute.

For Example: The address can be further divided into house number, street number, city, state, country and pin code, the name can also be divided into first name middle name and last name.



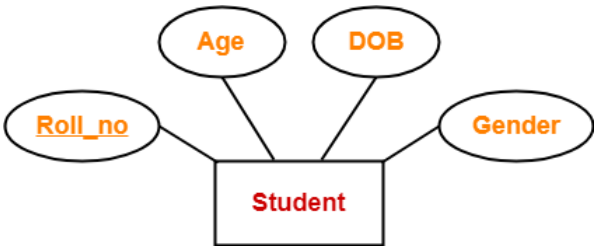
c) **Derived Attribute:** Derived Attributes are those attributes whose values are derived from the already stored attributes in the database.

For example: Age of a student calculated with current date and date of birth.



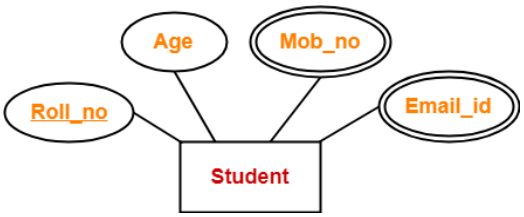

d) **Single Valued Attribute:** Single valued attribute are those which contains only a single specified value or a single unique value.

For example: Roll no, Aadhar Number, SSN No, etc.



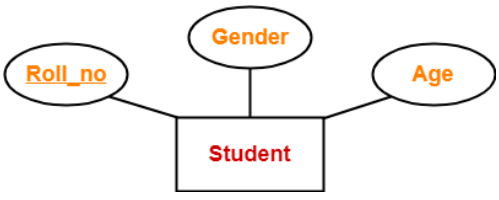

e) **Multi-Valued Attribute:** Multi-valued attributes are those which can have multiple values against them.

For example: Address(permanent, present), Email_id, Phone number of a student (Landline and mobile.




f) **Key Attributes:** Key attributes are those attributes which can identify an entity uniquely in an entity set.

For example: Here, the attribute “Roll_no” is a key attribute as it can identify any student uniquely.



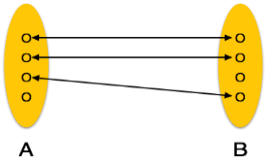
g) **Null Value Attribute:** Null value attributes are those which can be left blank without passing any value.

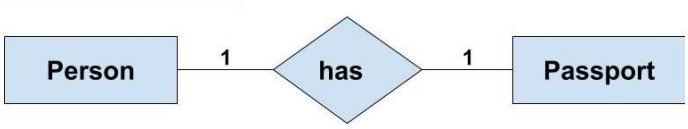
For example: A person can have a name without a middle name and hence middle name field can be left blank or null.

First Name + _____ + Last Name

C. **Relationship:** The association among entities is called a relationship.

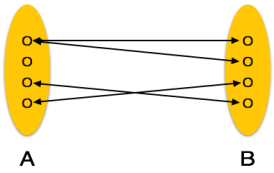
a) **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.

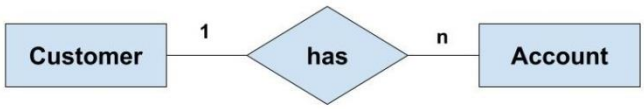



For example, If there are two entities 'Person' (Id, Name, Age, Address) and 'Passport' (Passport_id, Passport_no). So, each person can have only one passport and each passport belongs to only one person.

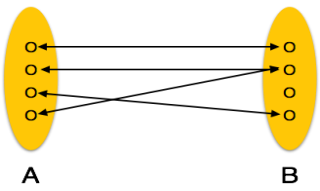
b) **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.

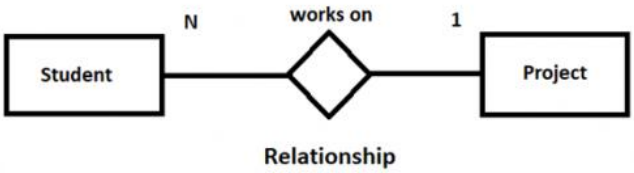



For example, If there are two entity type 'Customer' and 'Account' then each 'Customer' can have more than one 'Account' but each 'Account' is held by only one 'Customer'.

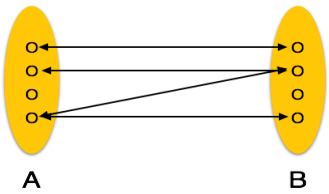
c) **Many-to-one** – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



A project can have more than one student working on it. A team of five students in a college in assigned a project that they need to complete in let us say one month. This states a relationship between two entities **Student** and **Project**.

d) **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.





Example: If there are two entity type 'Customer' and 'Product' then each customer can buy more than one product and a product can be bought by many different customers.

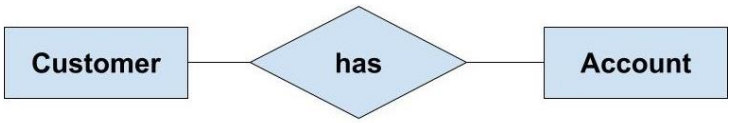
**Cardinality:**

It defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

**Degree of Relationship:**

The degree of a relationship is the number of entity types that participate (associate) in a relationship. By seeing an E-R diagram, we can simply tell the degree of a relationship i.e the number of an entity type that is connected to a relationship is the degree of that relationship.

For example: if we have two entity type 'Customer' and 'Account' and they are linked using the primary key and foreign key. We can say that the degree of relationship is 2 because here two entities are taking part in the relationship.




degree 2

**Relationships**

- When an Entity is related to another Entity, they are said to have a relationship.
- For example, A **Class** Entity is related to **Student** entity, becasue students study in classes, hence this is a relationship.
- Depending upon the number of entities involved, a **degree** is assigned to relationships.

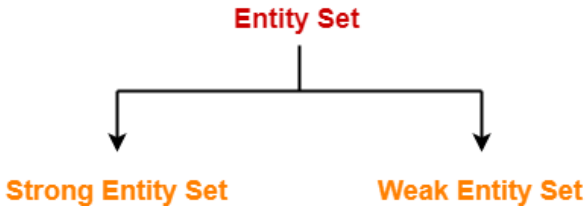
- For example, if 2 entities are involved, it is said to be **Binaryrelationship**, if 3 entities are involved, it is said to be **Ternary** relationship, and so on.

**Q: Explain about Classification of Entity Set in DBMS?**

An entity set is a group of entities that possess the same set of attributes. Each entity in an entity set has its own set of values for the attributes which make it distinct from other entities in a table. No two entities in an entity set will have the same values for the attributes.

**Types of Entity Sets-**

An entity set may be of the following two types-




**a) Strong Entity Set:**

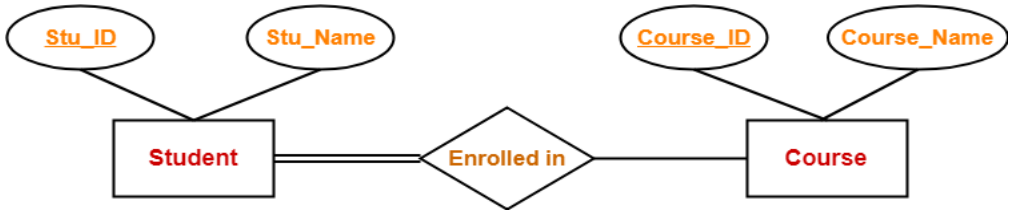
An entity set that has a primary key using which, entities in the table can be uniquely identified. This kind of entity set is termed as a strong entity set. it is also known as a regular entity set.

A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.In other words, a primary key exists for a strong entity set.

**Symbols Used:**

- Primary key of a strong entity set is represented by underlining it.
- A single rectangle is used for representing a strong entity set.
- A diamond symbol is used for representing the relationship that exists between two strong entity sets.
- A single line is used for representing the connection of the strong entity set with the relationship set.
- A double line is used for representing the total participation of an entity set with the relationship set.

**Example: Consider the following ER diagram-**




- Two strong entity sets "**Student**" and "**Course**" are related to each other.
- Student ID and Student name are the attributes of entity set "Student". Student ID is the primary key using which any student can be identified uniquely.
- Course ID and Course name are the attributes of entity set "Course".Course ID is the primary key using which any course can be identified uniquely.
- Double line between Student and relationship set signifies total participation.
- Single line between Course and relationship set signifies partial participation.

**b) Weak Entity Set-**

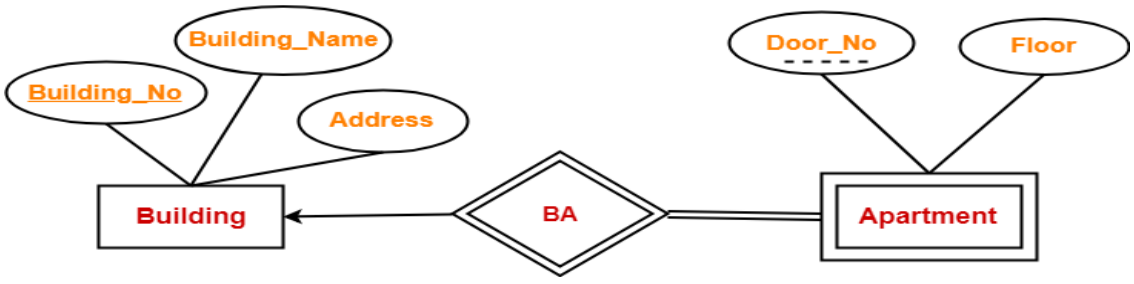
A weak entity set does not have any primary key which can identify each entity in a set distinctly.

In other words, a primary key does not exist for a weak entity set. However it contains a partial key called as a "discriminator". Discriminator can identify a group of entities from the entity set. Discriminator is represented by underlining with a dashed line.

**Symbols Used:**

- A **double rectangle** is used for representing a weak entity set.
- A **double diamond** symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as identifying relationship.
- A **double line** is used for representing the connection of the weak entity set with the relationship set.

**Example:**



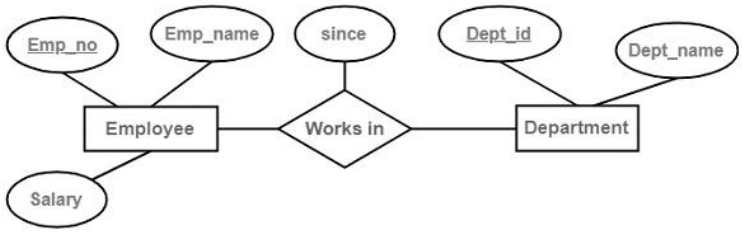
In this ER diagram,

- One strong entity set "**Building**" and one weak entity set "**Apartment**" are related to each other.
- Strong entity set "Building" has building number as its primary key.
- Door number is the discriminator of the weak entity set "Apartment".
- This is because door number alone cannot identify an apartment uniquely as there may be several other buildings having the same door number.
- Double line between Apartment and relationship set signifies total participation.
- Single line between Building and relationship set signifies partial participation.

## Q: How to Reduce (Conversion)an ER diagrams to tables?

Entity relationship diagram is the graphical representation of entities and relationships among those entities in the database.

**Example: consider this ER diagram to convert into tables**




Follow the steps given below for

the conversion of the ER diagrams to tables in the database management system (DBMS) –

**Step 1 – Conversion of strong entities**

- For each strong entity create a separate table with the same name.
- Includes all attributes, if there is any composite attribute divided into simple attributes and has to be included.
- Ignore multivalued attributes at this stage.
- Select the p key for the table.

Step 2 – **Conversion of weak entity**

- For each weak entity create a separate table with the same name.
- Include all attributes.

- Include the P key of a strong entity as foreign key is the weak entity.
- Declare the combination of foreign key and decimator attribute as P key from the weak entity.

Step 3 – **Conversion of one-to-one relationship**

- For each one to one relation, say A and B modify either A side or B side to include the P key of the other side as a foreign key.
- If A or B is having total participation, then that should be a modified table.
- If a relationship consists of attributes, include them also in the modified table.

Step 4 – **Conversion of one-to-many relationship**

- For each one to many relationships, modify the M side to include the P key of one side as a foreign key.
- If relationships consist of attributes, include them as well.

Step 5 – **Conversion of many-many relationship**

- For each many-many relationship, create a separate table including the P key of M side and N side as foreign keys in the new table.
- Declare the combination of foreign keys as P for the new table.
- If relationships consist of attributes, include them also in the new table.

Step 6 – **Conversion of multivalued attributes**

- For each multivalued attribute create a separate table and include the P key of the present table as foreign key.
- Declare the combination of foreign key and multivalued attribute as P keys.

### Table

After successful conversion, the result will be as follows –

| Emp_no | Dept_id | since |
|---|---|---|
| | | |

Schema : Works in ( Emp_no , Dept_id , since )

## Q: Explain about (EER) Enhanced entity-relationship model?

EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced entity-relationship diagrams are advanced database diagrams very similar to regular ER diagrams which represents requirements and complexities of complex databases.

**It is a diagrammatic technique for displaying the following concepts.**

A. Sub Class and Super Class

B. Specialization and Generalization

C. Union or Category

D. Aggregation

These concepts are used when they come in EER schema and the resulting schema diagrams called as EER Diagrams.

It is a diagrammatic technique for displaying the Sub Class and Super Class; Specialization and Generalization; Union or Category; Aggregation etc.

**A. Sub Class and Super Class**

- Sub class and Super class relationship leads the concept of Inheritance.
- Relationship between subclass and superclass is denoted with symbol. (d)

**Super Class:**

- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class. For example: Shape super class is having sub groups as Square, Circle, Triangle.

**Sub Class:**

- Sub class is a group of entities with unique attributes.
- Sub class inherits properties and attributes from its super class.

  For example: Square, Circle, Triangle are the sub class of Shape super class.

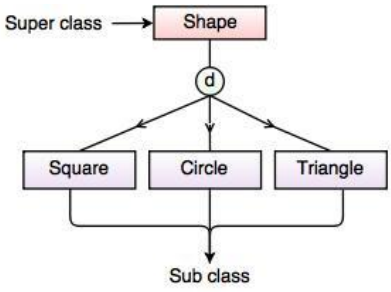



Fig. Super class/Sub class Relationship

**B. Specialization and Generalization**

**Generalization:**

- The process of defining a general entity type from a collection of specialized entity types.
- It is a bottom-up approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.

**For example:** Tiger, Lion, Elephant can all be generalized as Animals.

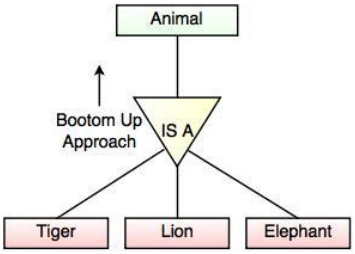



Fig. Generalization

**Specialization:**

- It is the opposite of generalization:
- Specialization is a process that defines a group entity which is divided into subgroups based on their characteristic.
- It is a top-down approach, in which one higher entity can be broken down into two lower-level entity.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

For example: Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

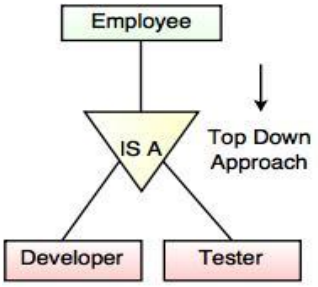



Fig. Specialization

**C. Category or Union**

- Category represents a Relationship of one super or sub class with more than one super class.

**For example** Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company. Category (sub class) →Owner is a subset of the union of the three super classes→Company, Bank, and Person.

A Category member must exist in at least one of its super classes.

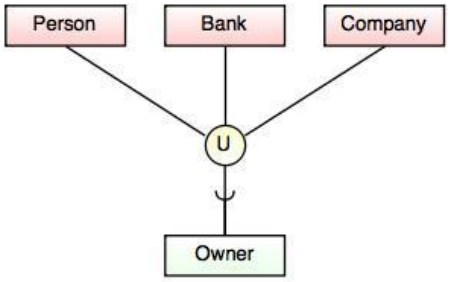



Fig. Categories (Union Type)

**D. Aggregation:**

- It is a process when two entities are treated as a single entity.
- Aggregation is a process that represents a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.

For Example: The relation between College and Course is acting as an Entity in Relation with Student.

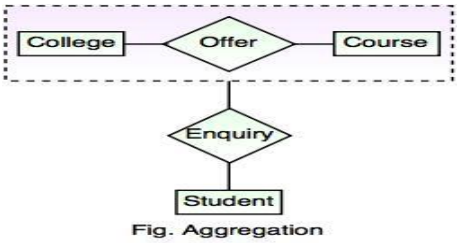



Fig. Aggregation

**Constraints on specialization and generalization:**

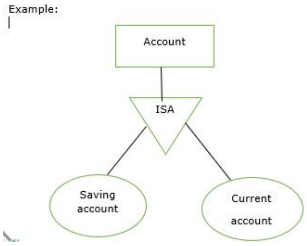
**Disjoint constraints**

Disjoint is nothing but intersection, the number of instances specified for the given superclass can participate in only one of the sub classes.

**Example:**

Account users can participate in saving account and current account but both are different so, it can be participated one at a time.

Example:




### Overlapping constraint

Two or more instances of the super class are participating in two or more sub classes then it is called overlapping constraints.

**Example**

A person who knows Java and PHP can participate in both teams.

### Complete constraint

Every instance participates in a relationship. All the instances of the superclass must participate in a relationship or into the sub class.

## Q: Explain about Advantages of ER Model?

The ER diagram is highly popular as it carries multiple benefits which include:

- **It is very simple:**

  ER model is very simple because if we know relationship between entities and attributes, then we can easily draw an ER diagram.

- **Blueprint to work:**

  It provides a blueprint to work with when you do start creating the actual database.

- **Highly Flexible:**

  The ER diagram can be effectively utilized for establishing and deriving relationships from the existing ones.

- **Better visual representation:**

  ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.

- **Effective communication tool:**

  It is an effective communication tool for database designer.

- **Highly integrated with relational model:**

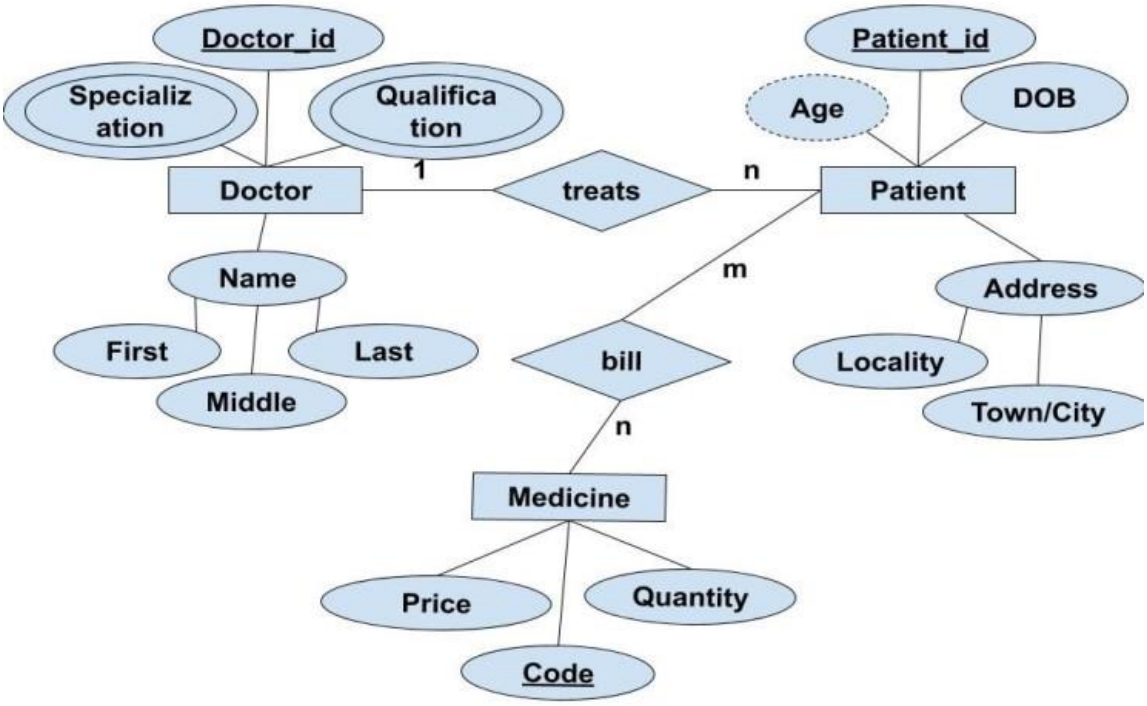
  ER model can be easily converted into relational model by simply converting ER model into tables.

- **Easy conversion to any data model:**

ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

**Q: Draw an ER-diagram for Hospital Management System:**

ER diagram for a hospital management system have three entities i.e Doctor, Patient, Medicine.




**Hospital Management System**

In the above diagram, Entity Doctor has key attribute 'doctor_id' which will be used to identify the doctors.

It also has two multivalued attributes as 'specialization' and 'qualification' as a doctor may have more than one qualification and may be specialized in more than one fields.

The Doctor and Patient entity have a one-to-many relationship as a Doctor may treat more than one patient. Similarly, Patient and Medicine have a many-to-many relationship as a patient may buy more than one medicine and vice-versa. '

Code' is the key attribute for Medicine which is unique for every medicine. The Patient has many attributes Patient_id, DOB, Age, etc. 'Age' is the derived attribute here. Also, it has a composite attribute 'Address' which can further be divided into two attributes 'Locality' and 'Town'.

**UNIT-III**

## What is Relational Model?

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

Some popular Relational Database management systems are:

DB2 and Informix Dynamic Server - IBM

Oracle and RDB – Oracle

SQL Server and Access - Microsoft

## Relational Model Concepts

a. **Attribute:** Each column in a Table. Attributes are the properties which define a relation.

   E.g: Student_Rollno, NAME,etc.

b. **Tables:** In the Relational model relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

c. **Tuple:** It is nothing but a single row of a table, which contains a single record.

d. **Relation Schema:** A relation schema represents the name of the relation with its attributes.

   **Ex:**Employee(Emp number, Name, Designation, Age, Salary)

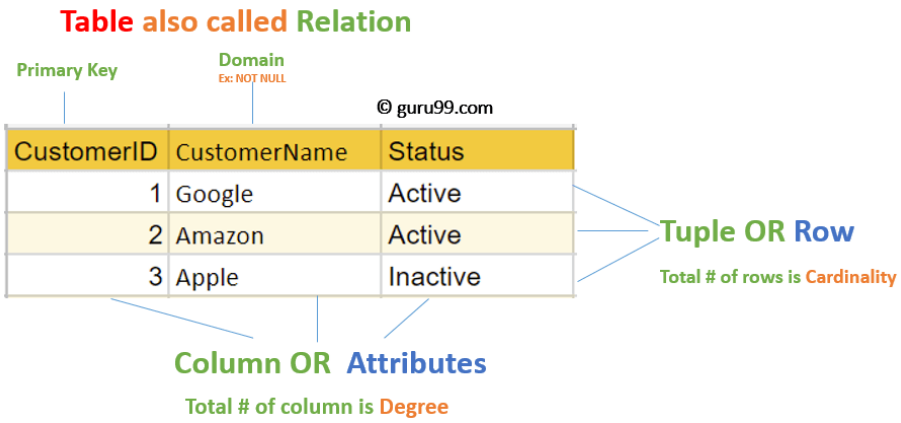
e. **Degree:** The total number of attributes which in the relation is called the degree of the relation.

f. **Cardinality:** Total number of rows present in the Table.

g. **Column:** The column represents the set of values for a specific attribute.

h. **Relation key:** The key for a table is the attribute that can uniquely identify all the tuples. In the Employee table, the key is Emp Number as it is unique for every single employee,.

i. **Attribute domain:** Every attribute has some pre-defined value and scope which is known as attribute domain




**Constraints:**

Every relation has some constraints that must hold for it to be called a relational model. These are as followed –

a) Key constraints - There must be at least one set of attributes that can identify a tuple in a unique manner. This set is known as a key.

b) Domain constraints - There are some domain specific constraints that must be followed in a database.

   i. Example - The salary of an employee cannot be negative so the salary field has only positive values.

c) Referential Integrity constraints - These constraints are used to describe the behaviour of foreign keys. A foreign key is a key of a relation that can be referred in another relation.

## Q: Explain about Codd's Rules ?

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

**Rule 1: Information Rule**

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

**Rule 2: Guaranteed Access Rule**

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value).

**Rule 3: Systematic Treatment of NULL Values**

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

**Rule 4: Active Online Catalog**

The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users.

**Rule 5: Comprehensive Data Sub-Language Rule**

A database can only be accessed using a languages that supports data definition, data manipulation, and transaction management operations.

**Rule 6: View Updating Rule**

All the views of a database, which can theoretically be updated, must also be updatable by the system.

**Rule 7: High-Level Insert, Update, and Delete Rule**

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

**Rule 8: Physical Data Independence**

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

**Rule 9: Logical Data Independence**

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it.

For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

**Rule 10: Integrity Independence**

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

**Rule 11: Distribution Independence**

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

This rule has been regarded as the foundation of distributed database systems.

**Rule 12: Non-Subversion Rule**

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

## Q: Explain about different Types of Keys in Relational Model?

**STUDENT**

| STUD_NO | STUD_NAME | STUD_PHONE | STUD_STATE | STUD_COUNTRY | STUD_AGE |
|---|---|---|---|---|---|
| 1 | RAM | 9716271721 | Haryana | India | 20 |
| 2 | RAM | 9898291281 | Punjab | India | 19 |
| 3 | SUJIT | 7898291981 | Rajsthan | India | 18 |
| 4 | SURESH | | Punjab | India | 21 |

**Table 1**

**STUDENT_COURSE**

| STUD_NO | COURSE_NO | COURSE_NAME |
|---|---|---|
| 1 | C1 | DBMS |
| 2 | C2 | Computer Networks |
| 1 | C2 | Computer Networks |

**Table 2**

### A. Candidate Key:

- Candidate Key The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD_NO in STUDENT relation.
- The value of Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation.
- The candidate key can be simple (having only one attribute) or composite as well.

For Example, STUD_NO is candidate key for relation STUDENT.

For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

**B. Super Key:**

- The set of attributes which can uniquely identify a tuple is known as Super Key.
- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

For Example, STUD_NO, (STUD_NO, STUD_NAME) etc.

**C. Primary Key:**

- The primary key is a column, or set of columns, whose values uniquely identify each row in the table.
- Each table in a relational database must be assigned a primary key.
- There can be more than one candidate key in relation out of which one can be chosen as the primary key.

For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

**D. Alternate Key:**

The candidate key other than the primary key is called an alternate key.

For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_PHONE will be alternate key (only one out of many candidate keys).

**E. Foreign Key:**

If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers.

The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute

For Example, STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

## Q: Explain about Relational Algebra?

**Relational algebra** is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform

this action. SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

Basic SQL Relational Algebra Operations

A. **Unary Relational Operations**

   SELECT (symbol: σ)

   PROJECT (symbol: π)

   RENAME (symbol: ρ)

B. **Relational Algebra Operations from Set Theory**

   UNION (υ)

   INTERSECTION ( ),

   DIFFERENCE (-)

   CARTESIAN PRODUCT ( x )

C. **Binary Relational Operations**

   JOIN

   DIVISION

**SELECT (σ):**

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition.

Select operator selects tuples that satisfy a given predicate.

The general form of selection operation is shown below. σ (Relation)

**Notation: σp(r)**

σ is the predicate

r stands for relation which is the name of the table

p is prepositional logic

**Example 1** $\sigma_{topic = "Database"}$ (Tutorials)

**Output** - Selects tuples from Tutorials where topic = 'Database'.

We can use operators such as =, <=, >=, >, ≠, etc. and also Boolean operators like and, or, and not

**Example2:**

| Studno | Name | Course | Year |
|---|---|---|---|
| cs1 | Kannan | Big Data | II |
| cs2 | Gowri Shankar | R language | I |
| cs3 | Lenin | Big Data | I |
| cs4 | Padmaja | Python Programming | I |

$\sigma_{course}$ = "Big Data" (STUDENT )

**output:**

| Studno | Name | Course | Year |
|---|---|---|---|
| cs1 | Kannan | Big Data | II |
| cs3 | Lenin | Big Data | I |

**Projection(π):**

- The projection method defines a relation that contains a vertical subset of Relation.
- This helps to extract the values of specified attributes to eliminates duplicate values.
- ***Π*** (pi) symbol is used to choose attributes from a relation. This operator helps you to keep specific columns from a relation and discards the other columns.

The general syntax for projection is shown below: π <attribute>(Relation)

**Example of Projection:**

| CID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |
| 4 | Alibaba | Active |

Here, the projection of CustomerName and status will give

$\Pi_{CustomerName, Status}$ (Customers)

| CustomerName | Status |
|---|---|
| Google | Active |
| Amazon | Active |
| Apple | Inactive |
| Alibaba | Active |

**Rename (ρ):**

- we may want to rename the relation name or attribute name or both relation name and attribute name. This is done with the help special operation **rename** operation.
- It is a unary operator
- The general notation is as follows

The general RENAME operation ρ can be expressed by any of the following forms:

- $\rho_{S(B1, B2, ..., Bn)}(R)$ changes both:
  - the relation name to S, *and*
  - the column (attribute) names to B1, B1, .....Bn
- $\rho_{S}(R)$ changes:
  - the *relation name* only to S
- $\rho_{(B1, B2, ..., Bn)}(R)$ changes:
  - the *column (attribute) names* only to B1, B1, .....Bn

**Union operation (∪)**

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

**Example:** $\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

**Output:** Projects the names of the authors who have either written a book or an article or both.

**Example:**

**Set Difference (-)**

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

- ✓ The attribute name of A has to match with the attribute name in B.
- ✓ The two-operand relations A and B should be either compatible or Union compatible.
- ✓ It should be defined relation consisting of the tuples that are in relation A, but not in B.

**Syntax: relation1 –relation2**

Course_1

| C_id | C_name |
|---|---|
| 11 | Foundation C |
| 21 | C++ |
| 31 | JAVA |

Course_2

| C_id | C_name |
|---|---|
| 12 | Python |
| 21 | C++ |

Course_1 - Course_2

| C_id | C_name |
|---|---|
| 11 | Foundation C |
| 31 | JAVA |

**Intersection**

- An intersection is defined by the symbol ∩

  Notation: A∩B

  Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.

Course_1

| C_id | C_name |
|---|---|
| 11 | Foundation C |
| 21 | C++ |
| 31 | JAVA |

Course_2

| C_id | C_name |
|---|---|
| 12 | Python |
| 21 | C++ |

Course_1 ∩ Course_2

| C_id | C_name |
|---|---|
| 21 | C++ |

**Cartesian Product(X)**

Cartesian Product in DBMS is an operation used to merge columns from two relations. It is also called Cross Product or Cross Join.

Assuming R and S as relations with n and m attributes respectively,
the Cartesian product R × S can be written as,
R (A1, A2, … An) × S (B1, B2, …, Bm)
Q = R × S can be written as, Q (A1, A2, … An,B1, B2, …, Bm)
Where, Degree (Q) = n+m
Count (Q) = Number of tuples in R * Number of tuples in S

R

| A | B | C |
|---|---|---|
| a1 | b1 | c3 |
| a2 | b1 | c5 |
| a3 | b4 | c7 |

S

| E | F |
|---|---|
| e1 | f1 |
| e2 | f5 |

R × S

| A | B | C | E | F |
|---|---|---|---|---|
| a1 | b1 | c3 | e1 | f1 |
| a1 | b1 | c3 | e2 | f5 |
| a2 | b1 | c5 | e1 | f1 |
| a2 | b1 | c5 | e2 | f5 |
| a3 | b4 | c7 | e1 | f1 |
| a3 | b4 | c7 | e2 | f5 |

**Join Operations:**

JOIN Operations: Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

**Types of JOIN:** Various forms of join operation are:

Inner Joins:

- ✓ Theta join
- ✓ EQUI join
- ✓ Natural join

Outer join:

- ✓ Left Outer Join
- ✓ Right Outer Join
- ✓ Full Outer Join

**Inner Join:**

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. They are classified into

**Natural Join (⋈):** Natural join (⋈) is a binary operator that is written as (R ⋈S)

where R and S are relations.

The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names.

R(X1, X2, ... Xm, Y1, Y2, ... Yn) and S (Y1, Y2, ... Yn, Z1, Z2, ... Zm)

For an example consider the tables **Employee** and **Dept** and their natural join:

*Employee*

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |

*Dept*

| DeptName | Manager |
|---|---|
| Finance | George |
| Sales | Harriet |
| Production | Charles |

*Employee ⋈ Dept*

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Harry | 3415 | Finance | George |
| Sally | 2241 | Sales | Harriet |
| George | 3401 | Finance | George |
| Harriet | 2202 | Sales | Harriet |

**Theta Join:** In Theta Join, we apply the condition on two relations and then only those selected rows based on condition are used in the cross product to be merged and included in the output.

The Theta Join of the two relations can be written as follows. R $\bowtie_{<x \square y>}$ S

*Car*

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

*Boat*

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

$Car \bowtie Boat$

$CarPrice \geq BoatPrice$

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

**Equijoin:** The most widely used join operation is equijoin. In equi-join, rows are joined on the basis of values of a common attribute between the two relations are equal. when θ is =, this type of θ-join is called Equijoin. The Equijoin of two relations can be written as follows.

$$R \bowtie_{<x=y>} S$$

Where x is an attribute of relation R and y is an attribute of relation S

and the value of Theta (θ) must be an equal to (=) operator

Example: The condition CarRent = BikeRent produces the pairs of rows which satisfy the condition.

Car

| CarModel | CarRent |
|---|---|
| CarA | 2000 |
| CarB | 5000 |
| CarC | 1000 |
| CarD | 3000 |

Bike

| BikeModel | BikeRent |
|---|---|
| BikeA | 3000 |
| BikeB | 1000 |
| BikeC | 900 |
| BikeD | 2000 |

Car $\bowtie_{<CarRent = BikeRent>}$ Bike

| CarModel | CarRent | BikeModel | BikeRent |
|---|---|---|---|
| CarA | 2000 | BikeD | 2000 |
| CarC | 1000 | BikeB | 1000 |

Left Outer Join (⟕): Left Outer Join is similar to a natural join, but it returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. Left Outer Join is also known as Left Join. Let R and S be two relations and the Left Outer Join (⟕) can be written as follows. R ⟕ S

Let R and S be two relations and the Left Outer Join (⟕) can be written as follows.

R ⟕ S

R

| A | B |
|---|---|
| a1 | b1 |
| a2 | b2 |

S

| B | C |
|---|---|
| b2 | c2 |
| b3 | c3 |

R ⟕ S

| A | B | C |
|---|---|---|
| a1 | b1 | null |
| a2 | b2 | c2 |

**Right Outer Join (⟖):** Right Outer Join is similar to a natural join, but it returns all the rows of the table on the right side

of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. Right Outer Join is also known as Right Join.

Let R and S be two relations and the Right Outer Join (⟖) can be written as follows.

R ⟖ S

**R**

| A | B |
|---|---|
| a1 | b1 |
| a2 | b2 |

**S**

| B | C |
|---|---|
| b2 | c2 |
| b3 | c3 |

**R ⟖ S**

| A | B | C |
|---|---|---|
| a2 | b2 | c2 |
| null | b3 | c3 |

**Full Outer Join (⟗):** Full Outer Join is similar to a natural join. It creates the result-set by combining result of both Left Outer Join and Right Outer Join. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values. Full Outer Join is also known as Full Join. Let R and S be two relations and the Right Outer Join (⟗) can be written as follows.

Let R and S be two relations and the Right Outer Join (⟗) can be written as follows.

R ⟗ S

**R**

| A | B |
|---|---|
| a1 | b1 |
| a2 | b2 |

**S**

| B | C |
|---|---|
| b2 | c2 |
| b3 | c3 |

**R ⟗ S**

| A | B | C |
|---|---|---|
| a1 | b1 | null |
| a2 | b2 | c2 |
| null | b3 | c3 |

**Division:** We define the division operation R/S as the set of all x values such that for every y value in S, there is a tuple (x, y) in R.

In other words,

$$R/S = \frac{R(x,\ y)}{S(y)} = R/S(x)$$

The following example will illustrate the division operation:

**Options**

| USN | SubID |
|---|---|
| 412 | CS175 |
| 412 | CS272 |
| 412 | CS351 |
| 413 | CS175 |
| 413 | CS272 |
| 532 | CS175 |
| 676 | CS272 |

**Elective1**

| SubID |
|---|
| CS175 |
| CS272 |

**Elective3**

| SubID |
|---|
| CS175 |
| CS272 |
| CS351 |

**Elective2**

| SubID |
|---|
| CS175 |

Result after Division operation:

| USN |
|---|
| 412 |
| 413 |

Options/Elective1

| USN |
|---|
| 412 |
| 413 |
| 532 |

Options/Elective2

| USN |
|---|
| 412 |

Options/Elective3

## Relational Calculus:

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms –

**A.** Tuple Relational Calculus (TRC)

**B.** Domain Relational Calculus (DRC)

## Tuple Relational Calculus (TRC):

Tuple relational calculus works on filtering the tuples based on the specified conditions. TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).This is also known as expression of relational calculus

**Syntax of TRC:** {T | Conditions)}

Where t is the resulting tuple (tuple variable), condition (t) is the conditional expression used to fetch t.

The result of such query is the set of all tuples t that satisfy condition (t).

**For example**, to find all employees whose salary is above Rs 50000, we can write the following tuple calculus expression:

{t | EMPLOYEE (t) AND t.Salary>50000}

The above statement selects all the tuples from EMPLOYEE relation such that tuples whose salary is greater than 50000.

**Domain Relational Calculus (DRC):**

The domain relational calculus works based on the filtering of the domain and the related attributes. DRC is the variable range over the domain elements or the filed values.

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Syntax of DRC in DBMS: {c1, c2,...,cn| F(c1, c2,... ,cn)}

The domain attributes in DRC can be represented as C1, C2,..., Cn and the condition related to the attributes can be denoted as the formula defining the condition for fetching the F(C1, C2, ...Cn )

Let us assume the same Product table in the database as follows:

| Product_id | Product Category | Product Name | Product Unit Price |
|---|---|---|---|
| 8 | New | TV Unit 1 | $100 |
| 10 | New | TV Unit 2 | $120 |
| 12 | Existing | TV Cabinet | $77 |

DRC for the product name attribute from the Product table needs where the product id is 10, It will be demoted as:

{< Product Name, Product_id> | ∈ Product ∧Product_id> 10}

The result of the domain relational calculus for the Product table will be

| Product_id | Product Name |
|---|---|
| 10 | TV Unit 2 |

Some of the commonly used logical operator notations for DRC are ∧ for AND,∨ for OR, and ¬ for NOT.

**Advantages and disadvantages of relational algebra:**

**1.Flexibility:** Different tables from which information has to be linked and extracted can be easily manipulated by operators such as project and join to give information in the form in which it is desired.

**2. Precision:** The usage of relational algebra and relational calculus in the manipulation of the relations between the tables ensures that there is no ambiguity, which may otherwise arise in establishing the linkages in a complicated network type database.

**Disadvantages**

**1.Performance:** If the number of tables between which relationships to be established are large and the tables themselves effect the performance in responding to the sql queries.

**2. Physical Storage** Consumption: With an interactive system an operation like join would depend upon the physical storage also.

## Q:Explain about Functional Dependencies ?

- A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets.
- It is denoted as **X** → **Y**, where X is a set of attributes that is capable of determining the value of Y.
- The attribute set on the left side of the arrow, **X** is called **Determinant**,
- while on the right side, **Y** is called the **Dependent**.
- Functional dependencies are used to mathematically express relations among database entities

| roll_no | name | dept_name | dept_building |
|---|---|---|---|
| 42 | abc | CO | A4 |
| 43 | pqr | IT | A3 |
| 44 | xyz | CO | A4 |
| 45 | xyz | IT | A3 |
| 46 | mno | EC | B2 |
| 47 | jkl | ME | B2 |

From the above table we can conclude some valid and invalid

**Example:roll_no→{ name, dept_name, dept_building }**

Here, roll_no can determine values of fields name, dept_name and dept_building, hence a **valid Functional dependency.**

**Example: name →dept_name**

Students with the same name can have different dept_name, hence this is **not a valid functional dependency.**

**Types of Functional dependencies in DBMS:**

Dependencies are classified into

1. Trivial functional dependency
2. Non-Trivial functional dependency
3. Multivalued functional dependency
4. Transitive functional dependency

**1. Trivial Functional Dependency**

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.

i.e. If **X →Y** and **Y is the subset of X**, then it is called trivial functional dependency

**For example,**

| roll_no | Name | age |
|---|---|---|
| 42 | Abc | 17 |
| 43 | Pqr | 18 |
| 44 | Xyz | 18 |

Here, **{roll_no, name} →name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}**

Similarly, **roll_no →roll_no** is also an example of trivial functional dependency.

**2. Non-trivial Functional Dependency**

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.

i.e. If **X →Y** and **Y is not a subset of X**, then it is called Non-trivial functional dependency.

| **roll_no** | name | age |
|---|---|---|
| 42 | Abc | 17 |
| 43 | Pqr | 18 |
| 44 | Xyz | 18 |

**For example,**

Here, **roll_no→name** is a non-trivial functional dependency,

since the dependent **name** is **not a subset of** determinant **roll_no**

Similarly, **{roll_no, name}** →**age** is also a non-trivial functional dependency, since **age** is **not a subset of {roll_no, name}**

**3. Multivalued Functional Dependency**

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other.**

i.e. If **a** →**{b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

**For example,**

| roll_no | Name | age |
|---|---|---|
| 42 | Abc | 17 |
| 43 | Pqr | 18 |
| 44 | Xyz | 18 |
| 45 | Abc | 19 |

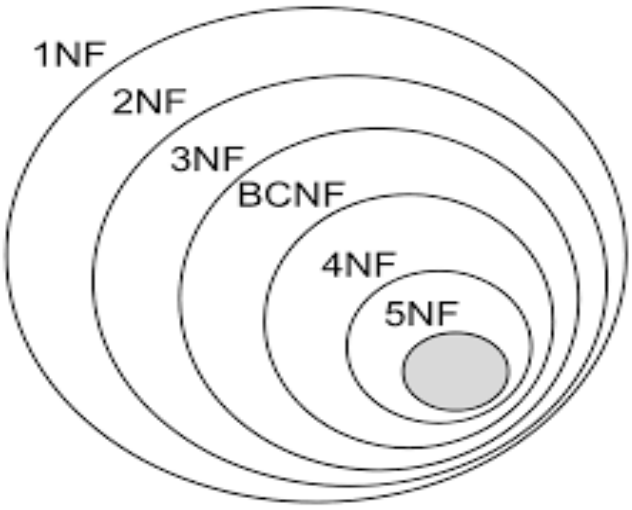
Here, **roll_no**→ **{name, age}** is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name** →**age** or **age** →**name doesn't exist !**)

**4. Transitive Functional Dependency**

In transitive functional dependency, dependent is indirectly dependent on determinant. i.e. If **a** →**b** & **b** →**c**, then according to axiom of transitivity, **a** →**c**. This is a **transitive functional dependency**

**Q:Explain about Normalization?**

**Normalization** is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables




**1. First Normal Form –**

a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

| id | NAME | COURSES |
|---|---|---|
| 1 | A | C1,C2 |
| 2 | E | C3 |
| 3 | M | C2,C3 |

In the above table Course is a multi-valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi-valued attribute

| ID | NAME | COURSE |
|---|---|---|
| 1 | A | C1 |
| 1 | A | C2 |
| 2 | E | C3 |
| 3 | M | C2 |
| 3 | M | C3 |

**2. Second normal form (2NF)**

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)

- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

**Example:** Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

| teacher_id | subject | teacher_age |
|---|---|---|
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

**Candidate Keys:** {teacher_id, subject}

**Non prime attribute:** teacher_age

The table is in 1 NF because each attribute has atomic values.

However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent o

n the proper subset of any candidate key of the table".

To make the table IN 2NF we can break it in two tables like this:

**teacher_details table:**

| teacher_id | teacher_age |
|---|---|
| 111 | 38 |
| 222 | 38 |
| 333 | 40 |

**teacher_details table**

| teacher_id | subject |
|---|---|
| 111 | Maths |
| 111 | Physics |
| 222 | Biology |
| 333 | Physics |
| 333 | Chemistry |

**teacher_subject table:**

Now the tables are in Second normal form (2NF).

**3. Third Normal Form (3NF)**

A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

.

| ID | NAME | SUBJECT | STATE | COUNTRY |
|---|---|---|---|---|
| 29 | Lalita | English | Gujrat | INDIA |
| 33 | Ramesh | Geography | Punjab | INDIA |
| 49 | Sarita | Mathematics | Maharashtra | INDIA |
| 78 | Zayed | History | Bihar | INDIA |

The candidate key in the above table is ID. The functional dependency set can be defined as ID->NAME, ID->SUBJECT, ID->STATE, STATE->COUNTRY.

If A->B and B->C are the two functional dependencies, then A->C is called the Transitive Dependency.

For the above relation, ID->STATE, STATE->COUNTRY is true. So we deduce that COUNTRY is transitively dependent upon ID. This does not satisfy the conditions of the Third Normal Form. So in order to transform it into Third Normal Form, we need to break the table into two tables in total and we need to create another table for STATE and COUNTRY with STATE as the primary key.

Below are the tables after normalization to the Third Normal Form.

**TEACHER_DETAILS:**

| ID | NAME | SUBJECT | STATE |
|---|---|---|---|
| 29 | Lalita | English | Gujrat |
| 33 | Ramesh | Geography | Punjab |
| 49 | Sarita | Mathematics | Maharashtra |
| 78 | Zayed | History | Bihar |

**STATE_COUNTRY:**

| STATE | COUNTRY |
|---|---|
| Gujrat | INDIA |
| Punjab | INDIA |
| Maharashtra | INDIA |
| Bihar | INDIA |

**Boyce and Codd Normal Form (BCNF)**

**Boyce and Codd Normal Form** is a higher version of the Third Normal form. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency ( X $\rightarrow$Y ), X should be a super Key.

**Fourth Normal Form (4NF)**

A table is said to be in the Fourth Normal Form when,

1. It is in the Boyce-Codd Normal Form.
2. And, it doesn't have Multi-Valued Dependency.

## Unit-4

**Q: What is structured query language? Explain various SQL commands.**

**A: SQL (Structured Query Language):** It is the interface language between the user and the oracle database. The American National Standards Institute (ANSI) has accepted SQL as standard access language for Tableal Database Management Systems.

- IBM developed SQL in mid of 1970's.
- Oracle incorporated in the year 1979.
- SQL used by IBM/DB2 and DS Database Systems.
- SQL adopted as standard language for RDBS by ASNI in 1989.

**Rules of SQL Statement:**

- An SQL statement starts with a verb. This verb may have additional nouns and adjectives.
- Each verb is followed by a number of clauses.
- Each clause has one or more parameters.
- A space separates clauses within an SQL statement.
- A comma separates parameters with in a clause.
- A semicolon is used to terminate the SQL statement.

**Q:DATA TYPES:**

**CHAR (Size):** This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.

**VARCHAR (Size) / VARCHAR2 (Size):** This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.

**NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point)..

The precision (p) determines the number of total digits. Scale(s)is omitted then the default is zero.

Ex: percentage number(4,2) total digits:4,digits after point:2 i.e 98.56

**DATE:** This data type is used to represent date and time. The standard format is DD-MM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format.

**LONG:** This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII form.

**There are 5 constraints available in ORACLE:**

**1. NOT NULL:** When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

***Syntax:***

**CREATE TABLE**Table_Name(column_namedata_type(*size*) **NOT NULL,** );

***Example:***

**CREATE TABLE** student (sno**NUMBER(3)NOT NULL,** name **CHAR(10)**);

**2. UNIQUE:** The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

***Syntax:***

**CREATE TABLE**Table_Name(column_namedata_type(*size*) **UNIQUE, ....**);

***Example:***

**CREATE TABLE** student (sno**NUMBER(3) UNIQUE,** name **CHAR(10)**);

**3. CHECK:** Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

***Syntax:***

**CREATE TABLE**Table_Name(column_namedata_type(*size*) **CHECK(*logical expression*), ....**);

***Example:*** **CREATE TABLE** student (sno**NUMBER (3),** name**CHAR(10)**,class **CHAR(5),CHECK**(class **IN**('CSE','CAD','VLSI'));

**4. PRIMARY KEY:** A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables. A table contains primary key is known as Master Table.

- ✓ It must uniquely identify each record in a table.
- ✓ It must contain unique values.
- ✓ It cannot be a null field.
- ✓ It cannot be multi port field.
- ✓ It should contain a minimum no. of fields necessary to be called unique.

***Syntax:***

**CREATE TABLE**Table_Name(column_namedata_type(*size*) **PRIMARY KEY, ....**);

***Example:***

**CREATE TABLE** faculty (fcode**NUMBER(3) PRIMARY KEY,**fname **CHAR(10)**);

**5. FOREIGN KEY:** It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a **detail table**. The table that defines the primary key and is referenced by the foreign key is called the **master table**.

***Syntax:*** **CREATE TABLE**Table_Name(column_namedata_type(*size*)

**FOREIGN KEY**(column_name) **REFERENCES**table_name);

**6. default**: The default constraint is used to insert a default value into a column.

**CREATE TABLE** student (sno**NUMBER(3)NOT NULL,** name **CHAR(10)**, colz varchar2(20) default 'sgcsrc');

**Q:SQL COMMANDS:**

There are five types of SQL commands. They are:

1. **Data Definition Language (DDL)**
2. **Data Manipulation Language (DML)**
3. **Data Retrieval Language (DRL)**

**4. Transational Control Language (T.C.L)**

**5. Data Control Language (D.C.L)**

**1. Data Definition Language (DDL):** It is used to define structure of a database.

- **a.** Create
- **b.** Alter
- **c.** Drop
- **d.** Rename

**a. create:** It is used to create a new table.

**Syn:** create table table_name(field_1 data_type(size),field_2 data_type(size),...);

**Ex:** SQL>create table student(sno number(3),sname char(10),class char(5));

**b. Alter:** It is used to modify or add fields to the structure of the table.

**(1)Alter Table ...Add...:**It is used to add some extra fields to the existing table.

**Syn:** alter table tablenameadd(new field_1 data_type(size), new field_2 data_type(size),..);

**ex:** sql>alter table std add(address char(10));

**(2)Alter Table...Modify...:**It is used to change the width as well as data type of the fields of the existing tables.

**Syn:** **alte**r table table_name**modify** (field_1 newdata_type(size), field_2 new data_type(size),.....field_newdata_type(size));

**ex:** sql>**alter** table student **modify**(sname varchar(10),class varchar(5));

**c. Drop table:** it is used to delete the structure of a table. it permanently deletes the records of the table.

**syn:** drop table table_name;

**ex:**sql>drop table student;

**d. rename:** it is used to change the name of the existing database table.

**syn:** rename table old_table_name to new_table_name;

**ex:** sql>rename table student to std;

**2. Data Manipulation Language (DML):** It is used to manipulate the data of a table.

- **A. Insert**
- **B. Update**
- **c. Delete**

**a. INSERT INTO:** It is used to add records to the table.

**i) Inserting a single record:**

**Syn:** insert into table_name values (data_1,data_2,..data_n);

**ex:** sql>insert into student values (1,'ravi','b.sc','palakol');

**ii) Inserting multiple records:**

**Syn:** insert into table name values(&data_1,&data_2,.....,&data_n);

**ex:** sql>insert into student values (&sno,'&sname','&class','&address');

Enter value for sno: 101

Enter value for name: Hemanth

Enter value for class: B.Sc

Enter value for name: Palakol

**iii) Inserting all values**

***Syntax:*** insert into relation_name_1 values(value1,valu2,...valuen);

***example:*** sql>insert into student(101,'rashmitha','mca','therlam');

**2. UPDATE-SET-WHERE:** This is used to update the content of a record in a table.

**Syn:** update table_name set field_name = data where field_name=data;

**ex:** sql>update student set sname = 'kumar' where sno=1;

**3. DELETE-FROM**: this is used to delete all the records of a table.

**a) delete-from**: this is used to delete all the records of table.

**syn:** delete from table_name;

**ex:** sql>delete from student;

**b) delete -from-where** : this is used to delete a selected record from a table.

**syn:** delete from table_name where condition;

**ex:** sql>delete from student where sno = 2;

**3. DRL(DATA RETRIEVAL LANGUAGE):** It is used to retrieve the data from the table.

**a. select from:** it displays all records of all fields.

**syn:** select * from table_name;

**ex:** sql> select * from dept;

| deptno | dname | loc |
|---|---|---|
| -------- | ---------- | ---------- |

| | | |
|---|---|---|
| 10 | accounting | new york |
| 20 | research | dallas |
| 30 | sales | chicago |
| 40 | operations | boston |

**b. SELECT FROM:** To display a set of fields for all records of table.

**Syn:** SELECT a set of fields FROM table_name;

**Ex:** SQL> select deptno, dname from dept;

| DEPTNO | DNAME |
|---|---|
| 10 | accounting |
| 20 | research |
| 30 | sales |
| 40 | operations |

**c.SELECT - FROM -WHERE:** This query is used to display a selected set of fields for a selected set of records of a table.

**Syn:** SELECT a set of fields FROMtable_nameWHERE condition;

**Ex:** sql> select * from dept where deptno<=20;

| deptno | dname | loc |
|---|---|---|
| 10 | accounting | new york |
| 20 | research | dallas |

**d. SELECT - FROM -GROUP BY:** This query is used to group all the records in a table together for each and every value of a specific key(s) and then display them for a selected set of fields in the table.

**Syn:** select a set of fields from table_name group by field_name;

**ex:** sql> select empno, sum (salary) from emp group by empno;

| empno | sum (salary) |
|---|---|
| 1 | 3000 |
| 2 | 4000 |
| 3 | 5000 |

4 6000

4rows selected.

**e. SELECT - FROM -ORDER BY:** This query is used to display a selected set of fields from a table in an ordered manner base on some field.

**Syn:** select a set of fields from table_name order by field_name;

**ex:** sql> select empno,ename,job from emp order by job;

| empno | ename | job |
|---|---|---|
| 4 | ravi | manager |
| 2 | aravind | Manager |
| 1 | sagar | clerk |
| 3 | Laki | clerk |

4rows selected.

**4. TRANSATIONAL CONTROL LANGUAGE(TCL):** It is used to make the transactions permanent.

**a. commit:** this command is used to end a transaction and made them permanent.

**syn:**sql>commit;

**ex:** sql>commit;

**b. save point:** save points are like marks to divide a very lengthy transaction to smaller once. they are used to identify a point in a transaction to which we can latter role back.

**syn:** sql>save point id;

**ex:** sql>save point xyz;

**c. role back:** a role back command is used to undo the current transactions.

**syn:** 1. role back( current transaction can be role back)

2. role back to save point id;

**ex:** sql>role back;

sql>role back to save point xyz;

**5. DATA CONTROL LANGUAGE (D.C.L):** DCL provides users with privilege commands

**a. grant:** the grant command allows granting various privileges to other users.

**syn:** sql>grant privileges on table_name to user_name;

**ex:** sql>grant select, update on emp to hemanth;

**b. revoke:** it is used to withdraw the privileges that has been granted to the user.

**syn:** sql>revoke privileges on table_name from user_name;

**ex:** sql>revoke select, update on emp from hemanth;

**Q: Explain about views in SQL.**

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it does not be stored in the database. View is a query that is stored as an object. It is a virtual table based on a select query.

**Creating Views**

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows –

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

***1. Example:***

SQL>CREATE VIEW employee AS SELECT empno,ename,jobFROM EMP

WHERE job = 'clerk';

View created.

SQL> SELECT * FROM EMPLOYEE;

| EMPNO | ENAME | JOB |
|---|---|---|
| 7369 | SMITH | CLERK |
| 7876 | ADAMS | CLERK |
| 7900 | JAMES | CLERK |
| 7934 | MILLER | CLERK |

**Characteristics of views:**

- We can use the name of the view anywhere in a SQL statement.
- Views are dynamically updated.

- Views provide a level of security in the database.
- Views may be used as the basis for reports.

**DROP VIEW:**This query is used to delete a view which is already created.

**Syn:** DROP VIEW View_name;

**Ex:** SQL> DROP VIEW EMPLOYEE;

View dropped

**Q:Explain SQL JOINs?**

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables –

**Table 1** – CUSTOMERS Table

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

**Table 2** – ORDERS Table

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
```

```
+-----+--------------------+------------+--------+
```

Now, let us join these two tables in our SELECT statement as shown below.

```
SQL> SELECT ID, NAME, AGE, AMOUNT
   FROM CUSTOMERS, ORDERS
WHERE  CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL −

- INNER JOIN − returns rows when there is a match in both tables.
- LEFT JOIN − returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN − returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN − returns rows when there is a match in one of the tables.
- SELF JOIN − is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN − returns the Cartesian product of the sets of records from the two or more joined tables

## UNIT V

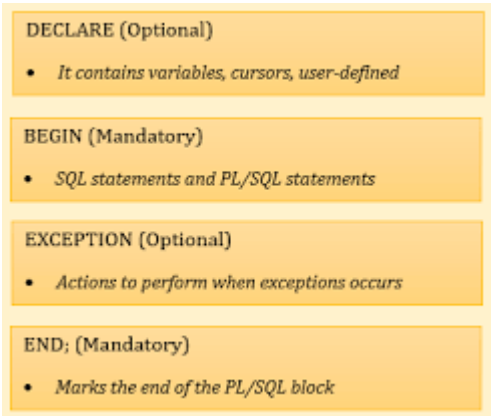
## PL/SQL

### Introduction

- PL/SQL stands for Procedural Language/Structured Query Language, which is provided by Oracle as a procedural extension to SQL.
- SQL is a declarative language. In SQL, the statements have no control to the program and can be executed in any order.
- But PL/SQL is a procedural language that makes up for all the missing elements in SQL.
- PL/SQL come up from the desire of programmers to have a language structure that was more familiar than SQL's purely declarative nature.

### Structure of PL/SQL

- PL/SQL is a block structured language. This means a PL/SQL program is made up of *blocks*, where block is a smallest piece of PL/SQL code having logically related statements and declarations. A block consists of three sections
- Declare,
- Begin,
- Exception
- End statement
- The different sections of PL/SQL block.




### Declare Section

Declare section declares the variables, constants, processes, functions, etc., tobe used in the other parts of program. It is an *optional* section.

### Begin Section

It is the executable section. It consists of a set of SQL and PL/SQL statements,which is executed when PL/SQL block runs. It is a compulsory section.

### Exception Section

This section handles the errors, which occurs during execution of the PL/SQLblock. Thissection allows the user to define his/her own error messages. Thissection executes only when an error occurs. It is an *optional* section.

### End Section

This section indicates the end of PL/SQL block.Every PL/SQL program must consist of at leastone block, which mayconsist of any number of nested sub-blocks. Figure 5.1 shows a typicalPL/SQLblock.

**PL/SQL Language Elements**

basic elements of PL/SQL language. Like other programming languages PL/SQL also have specific character sets, operators, indicators, punctuations, identifiers, comments, etc. In the following sections we will discuss about various language elements of PL/SQL.

1. **character sets**

A PL/SQL program consists of text having specific set of characters. 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.
- **Note:**PL/SQL is not case sensitive

2. **Lexical Units**

A line of PL/SQL program contains groups of characters known as lexicalunits, which can be classified asfollows:

- Delimiters
- Identifiers
- Literals
- Comments

**2.1 Delimiters**

A delimiter is a simple or compound symbol that has a special meaning toPL/SQL. Simple symbol consists ofone character, while compound symbolconsists of more than one character.

For example, to perform theadditionandexponentiation operation in PL/SQL, simple symbol delimiter + and compoundsymbol delimiter ** is used, respectively. PL/SQL supports following

simple symbol delimiters:

+ - * / = > <; % _ , ( ) @ : __

Compound symbol delimiters legal in PL/SQL are as follows:

<>! = ~= ^= <= >= := ** .. // <<>>

In the following sections we will discuss about these delimiters.

**2.2Identifiers**

- Identifiers are used in the PL/SQL programs to name the PL/SQL programitems as constants, variables,cursors, cursor variables, subprograms, etc.
- Identifiers can consists of alphabets, numerals, dollar signs,underscores,and number signs only. Any other characters like hyphens, slashes, blankspaces, etc. are illegal.
- An identifier must begin with an alphabetic letteroptionally followed by one or more characters (permissiblein identifier).
- Anidentifier cannot contain more than 30 characters.

**Example**

A, A1 ,Share$price,e_mail,phone#

**2.3 Literals**

A literal is an explicitly defined character, string, numeric, or Boolean value,which is not represented by anidentifier.

Examples:

integer numeric literals : 100 006 -10 0 +10

A real literal: 0.0 -19.0 3.56219 +43.99 .6 7. -4.56

**a)Character Literals**

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

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

**b)String Literals**

A string literal is enclosed within single quotes and may consistof one or more characters.

**Ex:**

"Good Morning!"

"TATA INFOTECH LTD"

"04-MAY-00"

"$15,000,000"

**c) Boolean Literals**

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

**2.4Comments**

Comments are used in the PL/SQL program to improve the readability andunderstandability of a program. A comment can appear anywhere in the programcode. The compiler ignores comments. A PL/SQL commentmaybe a single-line ormultiline.

**A)Single-Line Comments**

Single-line comments begin with a double hyphen (- -) anywhere on a line andextend to the end of the line.

**Ex:**

```
-- start calculations
```

**b) Multiline Comments**

Multiline comments begin with a slash-asterisk (/*) and end with an asteriskslash(*/), and can span multiple lines.

**Ex:**

```
/* Hello World! This is an example of multiline comments in PL/SQL */
```

**Q:Variables and Constants?**

Variables and constants can be used within PL/SQL block, in proceduralstatements and in SQL statements.These are used to store the values. Asthe program executes, the values of variables can change, but the valuesofconstants cannot. However, it is must to declare the variables and constants,before using these inexecutable portion of PL/SQL.

**Declaration**

Variables and constants are declared in the Declaration section of PL/SQLblock. These can be any of the SQLdata type like CHAR, NUMBER,DATE, etc.

**Variables Declaration**

The syntax for declaring a variable is as follows:

**Syn: identifier datatype;**

**Ex:**

*DECLARE*

Name VARCHAR2(10);

Age NUMBER(2);

Joining date DATE;

**Initializing the Variable**

By default variables are initialized to NULL at the time of declaration. If wewant to initialize the variable bysome other value, syntax would be as follows:

**Syn: Identifier datatype := value;**

**Or**

**Identifier datatype DEFAULT value;**

**Ex:**

Joining date DATE := 01-JULY-99; (or)

Joining date DATE DEFAULT 01-JULY-99;

**Declaring Constants**

Declaration of constant is similar to declaration of variable, except the keywordCONSTANT precedes thedatatype and it must be initialized by somevalue. The syntax for declaring a constant is as follows:

**Syn:**identifier CONSTANT datatype := value;

**Example**

To define the age limit as a constant, having value 30; the declaration statement would be as follows: Age limit CONSTANT NUMBER := 30;

**Operators Precedence**

The operations within an expression are done in a particular order depending on their precedence (priority). Table lists the operator's level of precedence from top to bottom. Operators listed in the same row have equal

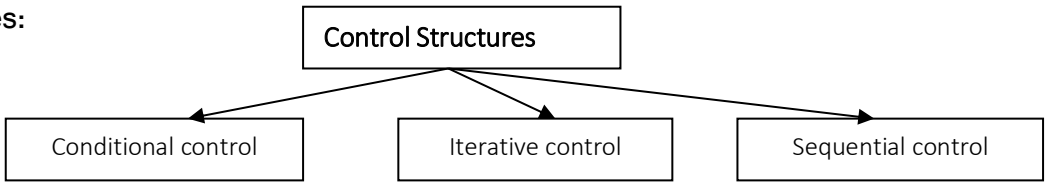
precedence.

Operators with higher precedence are applied first, but if parenthesesare used, expression within innermost parenthesis is evaluated first.

| Operator | operation |
|---|---|
| **, NOT | exponentiation, logical negation |
| +, - | identity, negation |
| *, / | multiplication, division |
| +, -, // | addition, subtraction concatenation |
| =, !=, <, >, <=, >=,IS NULL, LIKE BETWEEN, IN | comparison |
| AND | Conjunction |
| OR | Disjunction |

**Q:Control Structure in PL/SQL?**

Control Structure controls the flow of process. Control structure is broadly divided into three categories:




**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.

1. IF-THEN,
2. IF-THEN-ELSE,
3. IF-THEN-ELSEIF.

**IF-THEN**

It is the simplest form of IF condition. The syntax for this statement is as follows:

**Syn:**

**IF condition THEN**

**Sequence of statements**

**END IF;**

**Ex:**

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

**IF-THEN-ELSE**

If the condition is TRUE then the control executes the statement 1, other wise the control executes statement2

**Syn:**

**IF condition THEN**

**sequence of statements1**

**ELSE**

**sequence of statements2**

**END IF;**

**Ex:**

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

**IF-THEN-ELSIF**

**In this structure multiple conditions are involved**

**Syn:**

**IF condition1 THEN**

**sequence of statements1**

**ELSIF condition2 THEN**

**sequence of statements2**

**ELSE**

**sequence of statements3**

**END IF;**

**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 is the simplest form of iterative control. It encloses a sequence of statements between the keywords LOOP and END LOOP. The general syntax for LOOP control is as follows:

**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. The general syntax for this is as follows:

**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. The syntax for this is as follows:

**Syn:**

LOOP

EXIT WHEN condition

Sequence of statements

END LOOP

1. **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;

2. **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. The syntax is as follows:

**Syn:**

FOR counter IN lower limit ..higher limit LOOP

sequence of statements

END LOOP;

The sequence of statements is executed once for each integer in the range. After every iteration, the loop counter is incremented.

**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.

The syntax is as follows:

**GOTO label;**

. . . . . .

. . . . . .

<<**label**>>

**Statement**

**Steps to Create a PL/SQL Program**

1. Open notepad file on oracle as shown in the syntax

Syn: SQL>edit filename;

2. Type the required PL/SQL program and save it.

3. Now exit from the notepad

4. Type the following syntax on sql prompt to execute the program

Syn: SQL>START filename;

**Q: What are Cursors?**

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the*active* set.

**There are two types of cursors in PL/SQL: Implicit, Explicit cursors**

***Implicit cursors:***

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed. Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations.

The cursor attributes available are **%FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.**

The status of the cursor for each of these attributes are defined in the below table.

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

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

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

In the above PL/SQL Block, the salaries of all the employees in the 'employee' table are updated. If none of the employee's salary are updated we get a message 'None of the salaries where

updated'. Else we get a message like for example, 'Salaries for 1000 employees are updated' if there are 1000 rows in 'employee' table.

Explicit Cursors

An **explicit cursor** is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

**The General Syntax for creating a cursor is as given below:**

CURSOR cursor_name IS select_statement;

- *cursor_name – A suitable name for the cursor.*
- *select_statement – A select query which returns multiple rows.*

How to use Explicit Cursor?

There are four steps in using an Explicit Cursor.

- **DECLARE** the cursor in the declaration section.
- **OPEN** the cursor in the Execution Section.
- **FETCH** the data from cursor into PL/SQL variables or records in the Execution Section.
- **CLOSE** the cursor in the Execution Section before you end the PL/SQL Block.

**How to access an Explicit Cursor?**

These are the three steps in accessing the cursor.

1) Open the cursor.

2) Fetch the records in the cursor one at a time.

3) Close the cursor.

**General Syntax to open a cursor is:**

OPEN cursor_name;

**General Syntax to fetch records from a cursor is:**

FETCH cursor_name INTO record_name;

OR

FETCH cursor_name INTO variable_list;

**General Syntax to close a cursor is:**

CLOSE cursor_name;

When a cursor is opened, the first row becomes the current row. When the data is fetched it is copied to the record or variables and the logical pointer moves to the next row and it becomes the

current row. On every fetch statement, the pointer moves to the next row. If you want to fetch after the last row, the program will throw an error. When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.

**Points to remember while fetching a row:**

- We can fetch the rows in a cursor to a PL/SQL Record or a list of variables created in the PL/SQL Block.
- If you are fetching a cursor to a PL/SQL Record, the record should have the same structure as the cursor.
- If you are fetching a cursor to a list of variables, the variables should be listed in the same order in the fetch statement as the columns are present in the cursor.

Lets Look at the example below**Example 1:**

```
DECLARE
    emp_recemp_tbl%rowtype;
CURSOR emp_cur IS
SELECT *
FROM
WHERE salary > 10;
BEGIN
OPEN emp_cur;
FETCH emp_cur INTO emp_rec;
dbms_output.put_line (emp_rec.first_name || '  ' || emp_rec.last_name);
CLOSE emp_cur;
END;
```

**What are Explicit Cursor Attributes?**

Oracle provides some attributes known as Explicit Cursor Attributes to control the data processing while using cursors. We use these attributes to avoid errors while accessing cursors through OPEN, FETCH and CLOSE Statements.

When does an error occur while accessing an explicit cursor?

a) When we try to open a cursor which is not closed in the previous operation.

b) When we try to fetch a cursor after the last operation.

These are the attributes available to check the status of an explicit cursor.

| Attributes | Return values | Example |
|---|---|---|
| %FOUND | TRUE, if fetch statement returns at least one row. | Cursor_name%FOUND |
| | FALSE, if fetch statement doesn't return a row. | |
| %NOTFOUND | TRUE, , if fetch statement doesn't return a row. | Cursor_name%NOTFOUND |
| | FALSE, if fetch statement returns at least one row. | |
| %ROWCOUNT | The number of rows fetched by the fetch statement | Cursor_name%ROWCOUNT |
| | If no row is returned, the PL/SQL statement returns an error. | |
| %ISOPEN | TRUE, if the cursor is already open in the program | Cursor_name%ISNAME |
| | FALSE, if the cursor is not opened in the program. | |

**Q: Explain Procedures in PL/SQL?**

- 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 and Functions**

- Stored procedures and functions have many benefits
- It modifies one routine to affect multiple applications.
- It modifies one routine to eliminate duplicate testing.
- It ensures that related actions are performed together, or not at all, bydoing the activity through asingle path.
- It avoids PL/SQL parsing at runtime by parsing at compile time.
- It reduces the number of calls to the database and database network trafficby bundling the commands.

**Defining and Creating Procedures**

A procedure consists of two parts:

1. Specification
2. body.

**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.

**Body:**

- The procedure body starts with the keyword IS and ends with keyword END.

The procedure body is further subdivided into three parts:

1. Declarative part which consists of local declarations placed between keywords IS and BEGIN.

2. Executable part, which consists of actual logic of the procedure, included between keywords BEGIN and EXCEPTION.

3. Error/Exception handling part, an optional part placed between EXCEPTIONand END.

**The syntax for creating a procedure is follows:**

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

**Here**

**Create:** Creates a new procedure, if a procedure of same name alreadyexists, it gives an error.

**Replace:** Creates a procedure, if a procedure of same name already exists,it replace the older one by the newprocedure definition.

**Argument:** It is the name of the argument to the procedure.

**IN:** Specifies that a value for the argument must be specified when callingthe procedure.

**OUT:** Specifies that the procedure pass a value for this argument back toits calling environment afterexecution.

**IN OUT:** Specifies that a value for the argument must be specified whencalling the procedure and that theprocedure passes a value for this argumentback to its calling environment after execution. If no value isspecified then ittakes the default value IN.

**Datatype:** It is the unconstrained datatype of an argument. It supportsany data type supported by PL/SQL.No constraints like size constraints orNOT NULL constraints can be imposed on the data type. However, youcanput on the size constraint indirectly.

**Ex:**

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
dbms_output.put_line('Hello World!');
END;
/
```

**Executing/Invoking a Procedure**

The syntax used to execute a procedure depends on the environment from which the procedure is being called. From within SQLPLUS, a procedure can be executed by using the EXECUTE or EXEC command, followed by the procedure name. Any arguments to be passed to the procedure must be enclosed in parentheses following the procedure name.

**Syn:** EXEC procedurename(paramaters);

**Ex**: EXEC greetings;

**Output:** HelloWorld

**Removing a Procedure**

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

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

**Q: Explain about Functions?**

A Function is similar to procedure except that it must return one and only one value to the calling program.

The exact syntax for defining a function is given below

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

- Thus a function has two parts: function specification and function body.
- The function specification begins with keyword FUNCTION and ends with RETURN clause which indicates the data type of the value returned by the function.
- Function body is enclosed between the keywords IS and END. Sometimes END is followed by function name, but this is optional.

**function body also is composed of three parts:**

- ✓ declarative part,
- ✓ executablepart,
- ✓ Error/exception handling part (optional).

**Removing a Function**

To remove a function, use following command:

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

**Parameters**

Parameters are the link between a subprogram code and the code calling thesubprogram.

**Parameter Modes**

Parameter modes define the behavior of formal parameters of subprograms. There are three types of parameter modes: IN, OUT, IN/OUT.

**IN Mode**

IN mode is used to pass values to the called subprogram. Inside the called subprogram, an IN parameter acts like a constant and hence it cannot be assigned a new value.

**OUT Mode**

An OUT parameter returns a value back to the caller subprogram.

**IN/OUT**

An IN/OUT parameter performs the duty of both IN parameter as well as OUT parameter.

**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.

**Q:Explain Packages in PL/SQL?**

PL/SQL Packages is schema object and collection of related data type (variables, constants), cursors, procedures, functions are defining within a single context. Package are device into two part,

- Package Specification
- Package Body

Package specification block you can define variables, constants, exceptions and package body you can create procedure, function, and subprogram.

**PL/SQL Specification:**This contain the list of variables, constants, functions, procedure names which are the part of the package. PL/SQL specification are public declaration and visible to a program.

**Syn:**

```
CREATE[OR REPLACE] PACKAGE package_name
IS|AS
[declaration ]
BEGIN
    [PROCEDURE]
    [FUNCTION ]
END[package_name];
```

**PL/SQL Body :** This contains the actual PL/SQL statement code implementing the logics of functions, procedures which are you already before declare in "Package specification".

**Syn:**

```
CREATE[OR REPLACE] PACKAGE BODY package_name
IS|AS
[declaration]
BEGIN
        [initialization_statement]
        [PROCEDURE]
        [FUNCTION  ]
    EXCEPTION
        WHEN built-in_exception_name_1 THEN
            User defined statement (action) will be taken;
    END;
```

**PL/SQL Package Example**

PL/SQL Package example step by step explain to you, you are create your own package using this reference example. We have emp1 table having employee information

**Package Specification Code**

Create Package specification code for defining procedure, function IN or OUT parameter and execute package specification program.

```
CREATEor REPLACE PACKAGE pkg1
    IS|AS
    PROCEDURE pro1
        (noin number, name out varchar2);
    FUNCTION fun1
        (noin number)
        RETURN varchar2;
    END;
/
```

**Package Body Code**

Create Package body code for implementing procedure or function that are defined package specification. Once you implement execute this program.

```
CREATEor REPLACE PACKAGE BODY pkg1
IS
        PROCEDUREpro1(noinnumber,info our varchar2)
                IS
                BEGIN
                        SELECT*INTO temp FROM emp1 WHEREeno=no;
                END;

        FUNCTIONfun1(noin number)return varchar2
                IS
                name varchar2(20);
                BEGIN
                        SELECTenameINTO name FROM emp1 WHEREeno=no;
                        RETURN name;
                END;
END;
/
```

**PI/SQL Program calling Package**

Now we have a one package pkg1, to call package defined function, procedures also pass the parameter and get the return result.

*pkg_prg.sql*

```
DECLARE
        no number :=&no;
        name varchar2(20);
BEGIN
        pkg1.pro1(no,info);
        dbms_output.put_line('Procedure Result');
        dbms_output.put_line(info.eno||'     '||
        info.ename||'   '||
```

```
        info.edept||'   '||
        info.esalary||' '||);
        dbms_output.put_line('Function Result');
        name := pkg1.fun1(no);
        dbms_output.put_line(name);
END;
/
```

**Output:**

```
SQL>@pkg_prg
no number &n=2
Procedure Result
2   marks jems   Program Developer   38K
Function Result
```

**Advantages of Packages**

- You can create package to **store all related** functions and procedures are grouped together into single unit called packages.
- Packages are reliable to **granting a privileges**.
- All function and procedure within a package can **share variable** among them.
- Packages are support **overloading** to overload functions and procedures.
- Packages are **improve the performance** to loading the multiple object into memory at once, therefore, subsequent calls to related program doesn't required to calling physically I/O.
- Package is **reduce the traffic** because all block execute all at once.

## Q: Explain about EXCEPTION HANDLING IN PL/SQL?

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly. When an exception occurs a messages which explains its cause is recieved. **PL/SQL Exception message consists of three parts.**

**1) Type of Exception**

**2) An Error Code**

**3) A message**

By Handling the exceptions we can ensure a PL/SQL block does not exit abruptly(unexpected termination).

**Structure of Exception Handling.**

The General Syntax for coding the exception section

WHEN ex_name1 THEN <actions>

**Using when others:** If no errors are matched then it enters into when others

```
DECLARE
  Declaration section
BEGIN
  Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
WHEN ex_name2 THEN
   -Error handling statements
WHEN Others THEN
  -Error handling statements
END;
```

'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.

**nested PL/SQL blocks like this.**

```
DELCARE
  Declaration section
BEGIN
                    DECLARE
          Declaration section
                              BEGIN
                                Execution section
```

EXCEPTION

Exception section

END;

EXCEPTION

Exception section

END;

In the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends abruptly with an error.

**Types of Exception.**

There are 3 types of Exceptions.

a) Named System Exceptions

b) Unnamed System Exceptions

c) User-defined Exceptions

**a) Named System Exceptions**

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as **Named System Exceptions**.

**For example:** NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

Named system exceptions are:

1) Not Declared explicitly,

2) Raised implicitly when a predefined Oracle error occurs,

3) caught by referencing the standard name within an exception-handling routine.

| Exception Name | Reason | Error Number |
|---|---|---|
| CURSOR_ALREADY_OPEN | When you open a cursor that is already open. | ORA-06511 |
| INVALID_CURSOR | When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened. | ORA-01001 |
| NO_DATA_FOUND | When a SELECT...INTO clause does not return any row from a table. | ORA-01403 |

| | | |
|---|---|---|
| TOO_MANY_ROWS | When you SELECT or fetch more than one row into a record or variable. | ORA-01422 |
| ZERO_DIVIDE | When you attempt to divide a number by zero. | ORA-01476 |

**For Example:** Suppose a NO_DATA_FOUND exception is raised in a proc, we can write a code to handle the exception as given below.

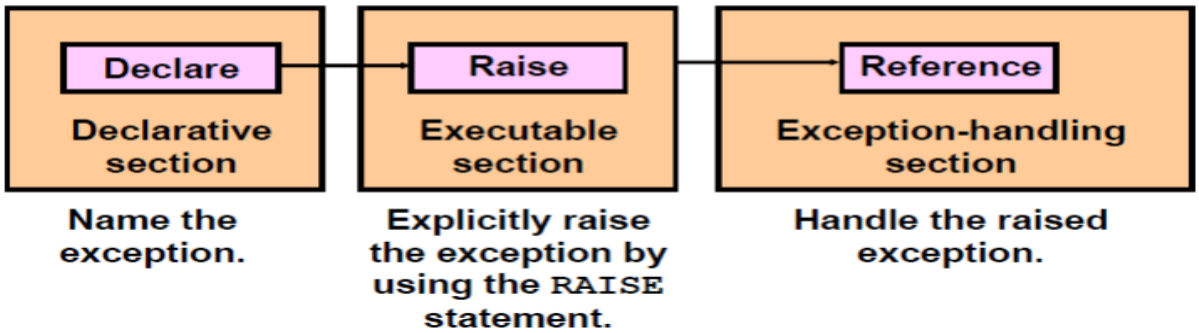
```
BEGIN
  Execution section
EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line ('A SELECT...INTO did not return any row.');
 END;
```

**b) Unnamed System Exceptions**

Those system exception for which oracle does not provide a name is known as unamed system exception. These exception do not occur frequently. These Exceptions have a code and an associated message.

**c) User-defined Exceptions**

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be **declared** and then raised explicitly, **using RAISE** statement.**There are following steps to define and handle user defined exceptions :**




**Steps to handle user defined exceptions**

**Step 1 : Declare the exception :**

To handle the exception first we have to declare the exception in declaration section.

```
my-exception EXCEPTION;
```

**Step 2 : RAISE exception :**

The second step is to RAISE exception using the RAISE keyword.

RAISE exception_name;

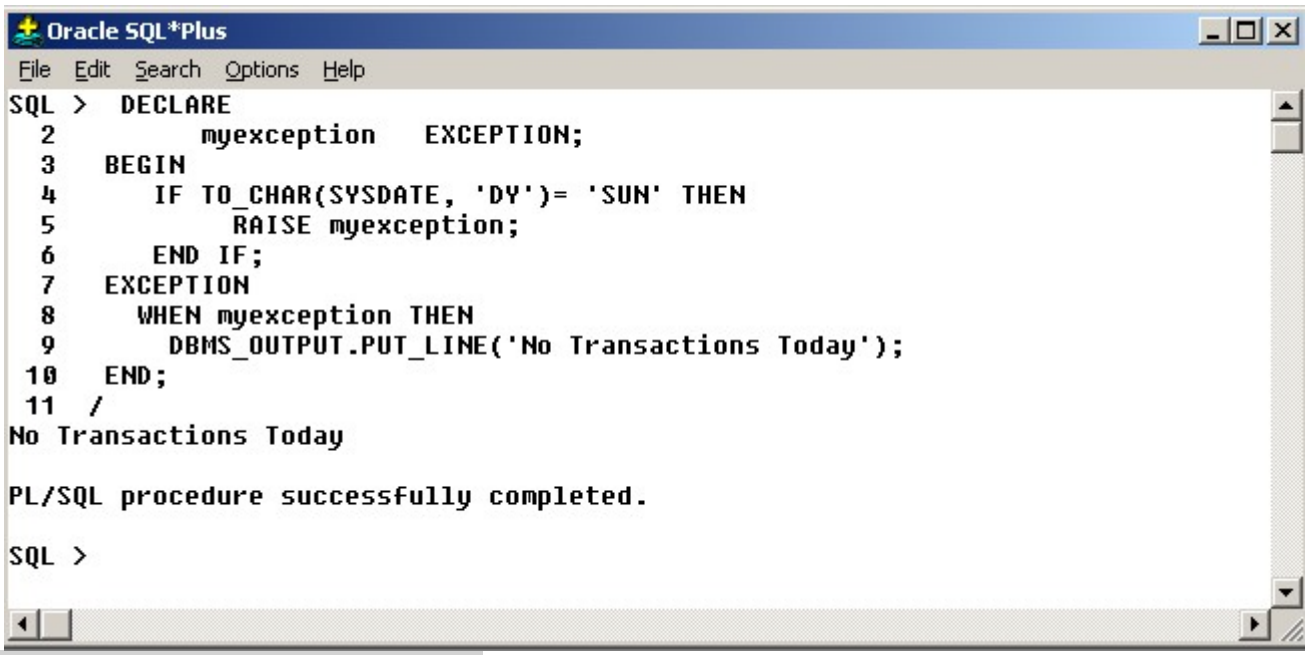
**Step 3 : Handle raised exception : we can handle the statements using following statement**

**WHEN** exception_name **THEN**

**Syntax to handle user defined:**

```
DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
  statement;
END;
```

**Example :**




**Q: Explain about Triggers in PL/SQL?**

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

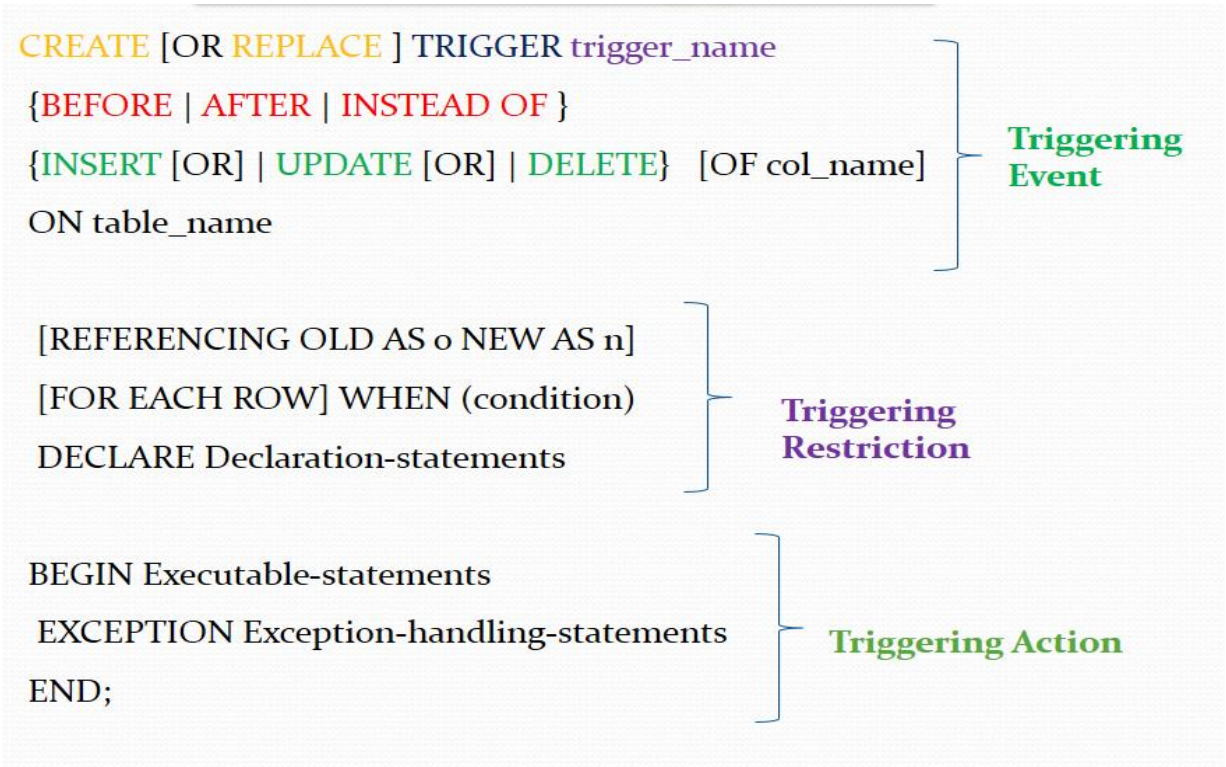
- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

- Triggers can be defined on the table, view, schema, or database with which the event is associated.

**Purpose of Triggers**

- To generate data automatically
- To enforce complex integrity constraints
- To customize complex security authorization
- To maintain replicate tables
- To audit data modifications.

**Creating Triggers:** Trigger creation consists of 3 parts




- CREATE [OR REPLACE] TRIGGER trigger_name – Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} – This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} – This specifies the DML operation.
- [OF col_name] – This specifies the column name that will be updated.
- [ON table_name] – This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] – This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.

- [FOR EACH ROW] – This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) – This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

**Types of PL/SQL Triggers**

There are two types of triggers based on the which level it is triggered.

**1) Row level trigger** - An event is triggered for each row upated, inserted or deleted.

**2) Statement level trigger** - An event is triggered for each sql statement executed.

PL/SQL Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.

**1)** BEFORE statement trigger fires first.

**2)** Next BEFORE row level trigger fires, once for each row affected.

**3)** Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.

**4)** Finally the AFTER statement level trigger fires.

**1) BEFORE UPDATE, Statement Level:** This trigger will insert a record into the table 'product_check' before a sql update statement is executed, at the statement level.

```
CREATE or REPLACE TRIGGER Before_Update_Stat_product
BEFORE
UPDATE ON product
Begin
INSERT INTO product_check
Values('Before update, statement level',sysdate);
END;
/
```

**2) BEFORE UPDATE, Row Level:** This trigger will insert a record into the table 'product_check' before each row is updated.

```
CREATE or REPLACE TRIGGER Before_Upddate_Row_product
BEFORE
UPDATE ON product
```

```
FOR EACH ROW
BEGIN
INSERT INTO product_check
Values('Before update row level',sysdate);
END;
/
```