# UNIT-1

# DATABASE

## 1.)What is Database?

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

Using the database, you can easily retrieve, insert, and delete the information.

**For example:** The college Database organizes the data about the admin, staff, students and faculty etc.

## Application and Uses of Database Management System (DBMS)

1.Railway Reservation System

Database is required to keep record of ticket booking, train's departure and arrival status. Also if trains get late then people get to know it through database update.

2.Library Management System

There are thousands of books in the library so it is very difficult to keep record of all the books in a copy or register. So DBMS used to maintain all the information relate to book issue dates, name of the book, author and availability of the book.

3.Banking

We make thousands of transactions through banks daily and we can do this without going to the bank. So how banking has become so easy that by sitting at home we can send or get money through banks. That is all possible just because of DBMS that manages all the bank transactions.

4.Universities and colleges

Examinations are done online today and universities and colleges maintain all these records through DBMS. Student’s registrations details, results, courses and grades all the information are stored in database.

5. Credit card transactions

For purchase of credit cards and all the other transactions are made possible only by DBMS. A credit card holder knows the importance of their information that all are secured through DBMS.

7.Social Media Sites

We all are on social media websites to share our views and connect with our friends. Daily millions of users signed up for these social media accounts like facebook, twitter, pinterest and Google plus. But how all the information of users are stored and how we become able to connect to other people, yes this all because DBMS.

8.Telecommunications

Any telecommunication company cannot even think about their business without DBMS. DBMS is must for these companies to store the call details and monthly post paid bills.

9.Finance

Those days have gone far when information related to money was stored in registers and files. Today the time has totally changed because there are lots f thing to do with finance like storing sales, holding information and finance statement management etc.

10.Military

Military keeps records of millions of soldiers and it has millions of files that should be keep secured and safe. As DBMS provides a big security assurance to the military information so it is widely used in militaries. One can easily search for all the information about anyone within seconds with the help of DBMS.

11.Online Shopping

Online shopping has become a big trend of these days. No one wants to go to shops and waste his time. Everyone wants to shop from home. So all these products are added and sold only with the help of DBMS. Purchase information, invoice bills and payment, all of these are done with the help of DBMS.

12.Human Resource Management

Big firms have many workers working under them. Human resource management department keeps records of each employee’s salary, tax and work through DBMS.

13.Manufacturing

Manufacturing companies make products and sales them on the daily basis. To keep records of all the details about the products like quantity, bills, purchase, supply chain management, DBMS is used.

14.Airline Reservation system

Same as railway reservation system, airline also needs DBMS to keep records of flights arrival, departure and delay status.

## 2.)PURPOSE OF A DATA BASE (or) need of DBMS?

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: **Storage of data** and **retrieval of data**.

**Storage:**

According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage.

Ex:
In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account. Let’s say bank stores saving account data at one place and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve in DBMS.

**Fast Retrieval of data**:

Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

## 3.)Database Management System

- Database management system is a software which is used to manage the database.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database.
- In the case of multiple users, it also maintains data consistency

For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications.

**DBMS allows users the following tasks:**

**1.Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.

**2.Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.

**3. Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.

**4.User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

## Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.
- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

## Advantages of DBMS

### Mass Storage

DBMS can store a lot of data in it. So for all the big firms, DBMS is really ideal technology to use. It can store thousands of records in it and one can fetch all that data whenever it is needed.

### 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.

**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.

**Data Protection**

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. DBMS gives a master level security to their data. No one can alter or modify the information without the privilege of using that data.

**Data Back up 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.

**Integrity**

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

**Platform Independent**

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

## Disadvantages of DBMS

**1.Cost of Hardware and Software:**

It requires a high speed of data processor and large memory size to run DBMS software.

**2.Size:**

It occupies a large space of disks and large memory to run them efficiently.

**3.Complexity:**

Database system creates additional complexity and requirements.

**4.Higher impact of failure:**

Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the  database is damaged due to electric failure or database corruption then the data may be lost forever.

## 4.)VIEWS OF DATA BASE

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction.

The three level of DBMS architecture, The top level of that architecture is "view level". The view level provides the "**view of data**" to the users and hides the irrelevant details such as data relationship, database schema, constraints, security etc from the user.

### 1. Data abstraction

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.

**Physical level**:

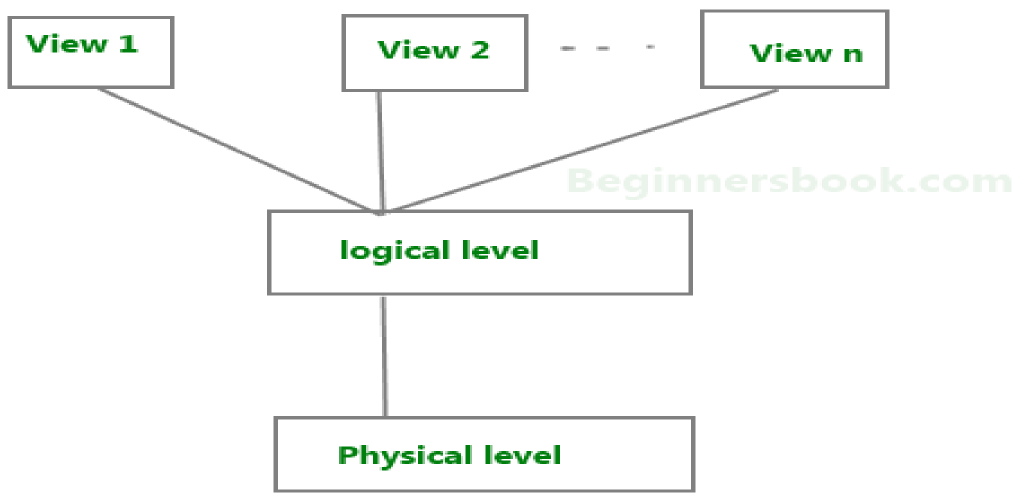
This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.
At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

**Logical level**:

This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database. At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems

**View level**:

Highest level of data abstraction. This level describes the user interaction with database system..At **view level**, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.



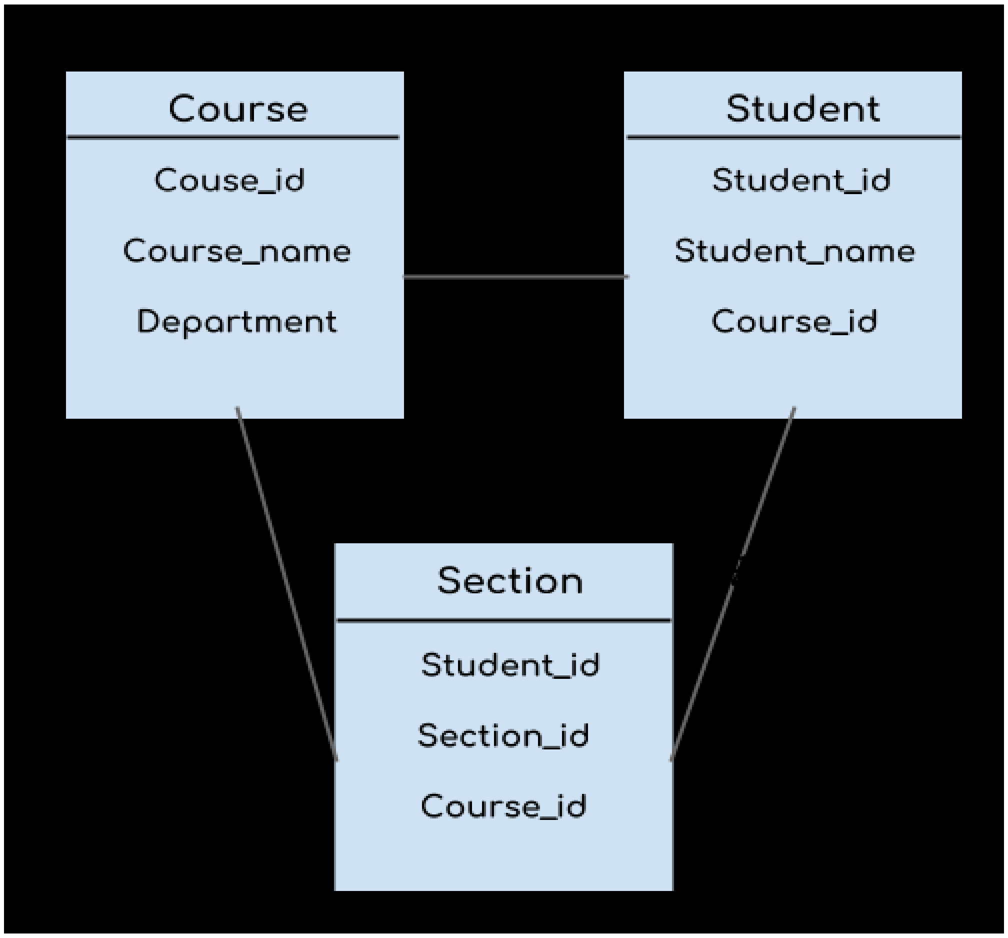
Three Levels of data abstraction

## 2.Instance and schema

**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.



1. 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.
2. Design of database at logical level is called **logical schema**, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).
3. Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

.

**DBMS 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.

## 5.)DATABASE LANGUAGE:

A DBMS has appropriate languages and interfaces to express database queries and updates. Database languages can be used to read, store and update the data in the database.

### Types of Database Language

#### 1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

## 2. Data Manipulation Language

**DML** stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.

**Explain Plan:** It has the parameter of explaining data.

**Lock Table:** It controls concurrency.

## 3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.The DCL execution is transactional. It also has rollback parameters.

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

## 4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

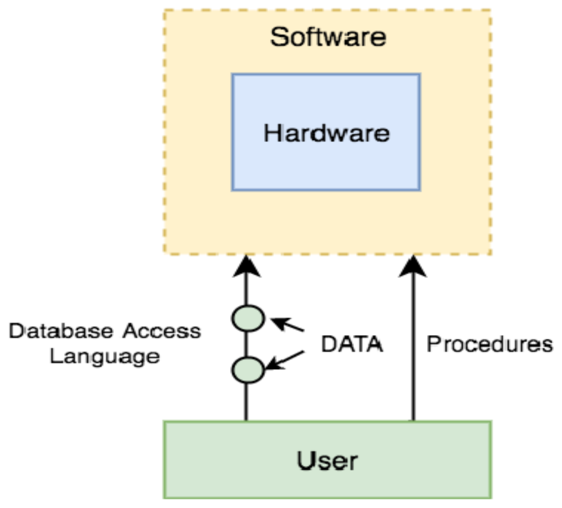
Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

## 6.)Components of DBMS

The database management system can be divided into five major components, they are:

1. Hardware
2. Software
3. Data
4. Procedures
5. Database Access Language



**Hardware**

When we say Hardware, we mean computer, hard disks, I/O channels for data, and any other physical component involved before any data is successfully stored into the memory.

When we run  MySQL on our personal computer, then our computer's Hard Disk, our Keyboard using which we type in all the commands, our computer's RAM, ROM all become a part of the DBMS hardware.

**Software**

This is the main component, as this is the program which controls everything. The DBMS software is more like a wrapper around the physical database, which provides us with an easy-to-use interface to store, access and update data.

The DBMS software is capable of understanding the Database Access Language and interpret it into actual database commands to execute them on the DB.

## Data

Data is that resource, for which DBMS was designed. The motive behind the creation of DBMS was to store and utilise data.

In a typical Database, the user saved Data is present and meta data is stored.

Metadata is data about the data. This is information stored by the DBMS to better understand the data stored in it.

## Procedures

Procedures refer to general instructions to use a database management system. This includes procedures to setup and install a DBMS. To login and logout of DBMS software, to manage databases, to take backups, generating reports etc.

## Database Access Language

Database Access Language is a simple language designed to write commands to access, insert, update and delete data stored in any database.

## User:

A user can write commands in the Database Access Language and submit it to the DBMS for execution, which is then translated and executed by the DBMS.

User can create new databases, tables, insert data, fetch stored data, update data and delete the data using the access language.

## 7.)<u>DATABASE MANAGEMENT SYSTEM & ITS FUNCTIONS</u>

- **Database Management System (DBMS)** refers to the technology solution used to optimize and manage the storage and retrieval of data from databases.
- DBMS offers a systematic approach to manage databases via an interface for users as well as workloads accessing the databases via apps. The management responsibilities for DBMS encompass information within the databases, the processes applied to databases (such as access and modification), and the database’s logic structure.
- DBMS also facilitates additional administrative operations such as change management, disaster recovery, compliance, and performance monitoring, among others.

Functions of, DBMS has the following key components:

**Software.**

DBMS is primarily a software system that can be considered as a management console or an interface to interact with and manage databases. The interfacing also spreads across real-world physical systems that contribute data to the backend databases. The OS, networking software, and the hardware infrastructure is involved in creating, accessing, managing, and processing the databases.

**Data.**

DBMS contains operational data, access to database records and metadata as a resource to perform the necessary functionality. The data may include files with such as index files, administrative information, and data dictionaries used to represent data flows, ownership, structure, and relationships to other records or objects.

**Procedures.**

While not a part of the DBMS software, procedures can be considered as instructions on using DBMS. The documented guidelines assist users in designing, modifying, managing, and processing databases.

**Database languages.**

These are components of the DBMS used to access, modify, store, and retrieve data items from databases; specify database schema; control user access; and perform other associated database management operations. Types of DBMS languages include Data Definition Language (DDL), Data Manipulation Language (DML), Database Access Language (DAL) and Data Control Language (DCL).

**Query processor.**

The query processor acts as an intermediary between users and the DBMS data engine in order to communicate query requests. When users enter an instruction in SQL language, the command is executed from the high-level language instruction to a low-level language that the underlying machine can understand and process to perform the appropriate DBMS functionality. In addition to instruction parsing and translation, the query processor also optimizes queries to ensure fast processing and accurate results.

**Runtime database manager.**

A centralized management component of DBMS that handles functionality associated with runtime data, which is commonly used for context-based database access. This component checks for user authorization to request the query; processes the approved queries; devises an optimal strategy for query execution; supports concurrency so that multiple users can simultaneously work on same databases; and ensures integrity of data recorded into the databases.

**Database manager.**

The database manager performs DBMS functionality associated with the data within databases. Database manager allows a set of commands to perform different DBMS operations that include creating, deleting, backup, restoring, cloning, and other database maintenance tasks. The database manager may also be used to update the database ..

**Database engine.**

This is the core software component within the DBMS solution that performs the core functions associated with data storage and retrieval. A database engine is also accessible via APIs that allow users or apps to create, read, write, and delete records in databases.

**Reporting.**

The report generator extracts useful information from DBMS files and displays it in structured format based on defined specifications. This information may be used for further analysis, decision making, or business intelligence.

## Benefits of DBMS

DBMS was designed to solve the fundamental problems associated with storing, managing, accessing, securing, and auditing data in traditional file systems. Traditional database applications were developed on top of the databases, which led to challenges such as data redundancy, isolation, integrity constraints, and difficulty managing data access.

**Data security.**

The DBMS system is also responsible to maintain optimum performance of querying operations while ensuring the validity, security and consistency of data items updated to a database.

**Data sharing.**

Fast and efficient collaboration between users.

**Data access and auditing.**

Controlled access to databases. Logging associated access activities allows organizations to audit for security and compliance.

**Data integration.**

Instead of operating island of database resources, a single interface is used to manage databases with logical and physical relationships.

**Abstraction and independence.**

Organizations can change the physical schema of database systems without necessitating changes to the logical schema that govern database relationships. As a result, organizations can upgrade storage and scale the infrastructure without impacting database operations. Similarly, changes to the logical schema can be applied without altering the apps and services that access the databases

**Uniform management and administration.**

A single console interface to perform basic administrative tasks makes the job easier for database admins and IT users.

## 8.)DATABASE DESIGN

Designing of database is most important responsibility of the software professionals who are dealing with the database related projects. For this they follow the Design Methodology. It helps the designer to plan, manage, control, and evaluate database development projects.

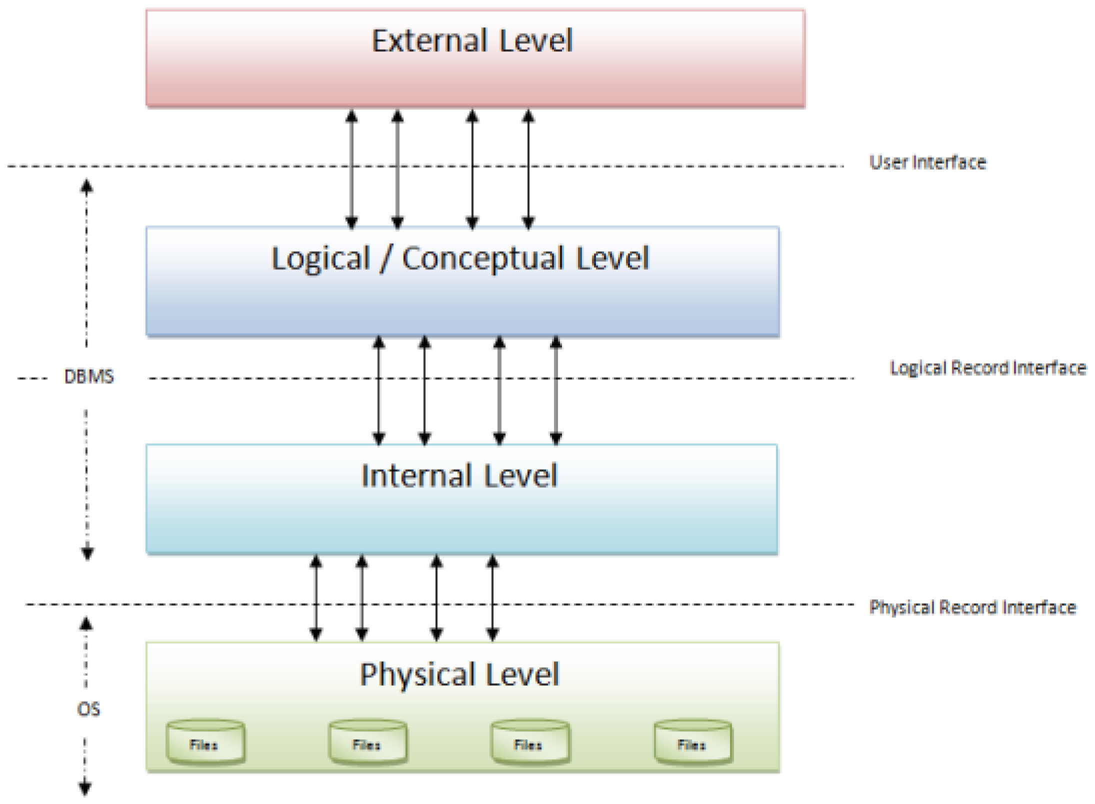
Design methodology:

- ➢ A structured approach that uses procedures, techniques, tools, and documentation aids to support and facilitate the process of design.
- ➢ A design methodology consists of phases each containing a number of steps, which guide the designer in the techniques appropriate at each stage of the project.

Phases of Design Methodology

The database design methodology is divided into three main phases. These are:

- Conceptual database design
- Logical database design
- Physical database design



## Conceptual database design

The process of constructing a model of the information used in an enterprise, independent of all physical considerations.

The conceptual database design phase begins with the creation of a conceptual data model of the enterprise, which is entirely independent of implementation details such as the target DBMS, application programs, programming languages, hardware platform, performance issues, or any other physical considerations.

## Logical database design

It is a process of constructing a model of the information used in an enterprise based on specific data model, but independent of a particular DBMS and other physical considerations.

The logical database design phase maps the conceptual model on to a logical model, which is influenced by the data model for the target database (for example, the relational model). The logical data model is a source of information for the physical design phase.

The output of this process is a global logical data model consisting of an Entity-Relationship diagram, relational schema, and supporting documentation that describes this model, such as a data dictionary.

## **Physical database design**

It is a description of the implementation of the database on secondary storage. it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

The physical database design phase allows the designer to make decisions on how the database is to be implemented.

For example, decisions taken during physical for improving performance, such as merging relations together, might affect the structure of the logical data model, which will have an associated effect on the application design.

One of the main objectives of physical database design is to store data in an efficient way. There are a number of factors that we may use to measure efficiency:

### **Transaction throughput**:

This is the number of transactions that can be processed in a given time interval. In some systems, such as airline reservations, high transaction throughput is critical to the overall success of the system.

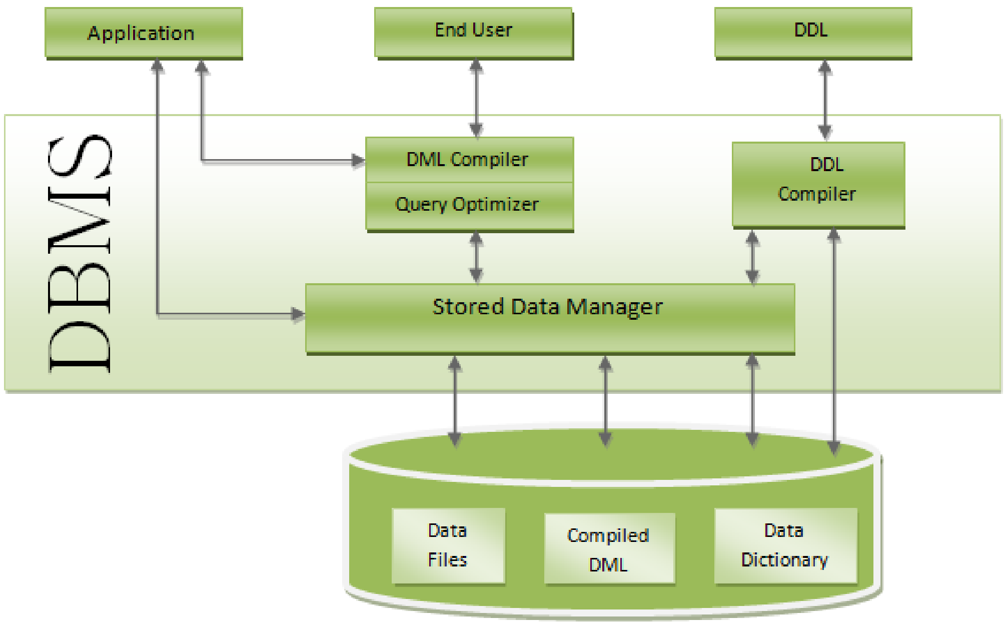
**<u>Disk storage</u>**:

This is the amount of disk space required to store the database files. The designer may wish to minimize the amount of disk storage used.

Response time

It is the time required for the completion of a single transaction. From a user's point of view, we want to minimize response time as much as possible.

## 9.)Structure of DBMS



### Applications:

It can be considered as a user friendly web page where the user enters the requests. Here he simply enters the details that he needs and presses buttons to get the data.

### End User:

They are the real users of the database. They can be developers, designers, administrator or the actual users of the database.

**DDL:**

Data Definition Language (DDL) is a query fired to create database, schema, tables, mappings etc in the database. These are the commands used to create the objects like tables, indexes in the database for the first time. In other words, they create structure of the database.

**DDL Compiler:**

This part of database is responsible for processing the DDL commands. That means these compiler actually breaks down the command into machine understandable codes. It is also responsible for storing the metadata information like table name, space used by it, number of columns in it, mapping information etc.

**DML Compiler:**

When the user inserts, deletes, updates or retrieves the record from the database, he will be sending request which he understands by pressing some buttons. But for the database to work/understand the request, it should be broken down to object code. This is done by this compiler. One can imagine this as when a person is asked some question, how this is broken down into waves to reach the brain!

**Query Optimizer:**

When user fires some request, he is least bothered how it will be fired on the database. He is not all aware of database or its way of performance. But whatever be the request, it should be efficient enough to fetch, insert, update or delete the data from the database.

The query optimizer decides the best way to execute the user request which is received from the DML compiler. It is similar to selecting the best nerve to carry the waves to brain!

**Stored Data Manager:**

This is also known as Database Control System. It is one the main central system of the database. It is responsible for various tasks

It converts the requests received from query optimizer to machine understandable form.  It makes actual request inside the database. It is like fetching the exact part of the brain to answer.

It helps to maintain consistency and integrity by applying the constraints.  That means, it does not allow inserting / updating / deleting any data if it has child entry. Similarly it does not allow entering any duplicate value into database tables.

It controls concurrent access. If there is multiple users accessing the database at the same time, it makes sure, all of them see correct data. It guarantees that there is no data loss or data mismatch happens between the transactions of multiple users.

It helps to backup the database and recover data whenever required. Since it is a huge database and when there is any unexpected exploit of transaction, and reverting the changes are not easy. It maintains the backup of all data, so that it can be recovered.

**Data Files:**

It has the real data stored in it. It can be stored as magnetic tapes, magnetic disks or optical disks.

**Compiled DML:**

Some of the processed DML statements (insert, update, delete) are stored in it so that if there is similar requests, it will be re-used.

**Data Dictionary:**

It contains all the information about the database. As the name suggests, it is the dictionary of all the data items. It contains description of all the tables, view, materialized views, constraints, indexes, triggers etc.

**10.)What is a Database Transaction?**

A transaction is a logical unit of processing in a DBMS which entails one or more database access operation. In a nutshell, database transactions represent real-world events of any enterprise.

All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.

**Database Transactions**

- A transaction is a program unit whose execution may or may not change the contents of a database.
- The transaction is executed as a single unit.
- If the database operations do not update the database but only retrieve data, this type of transaction is called a read-only transaction.

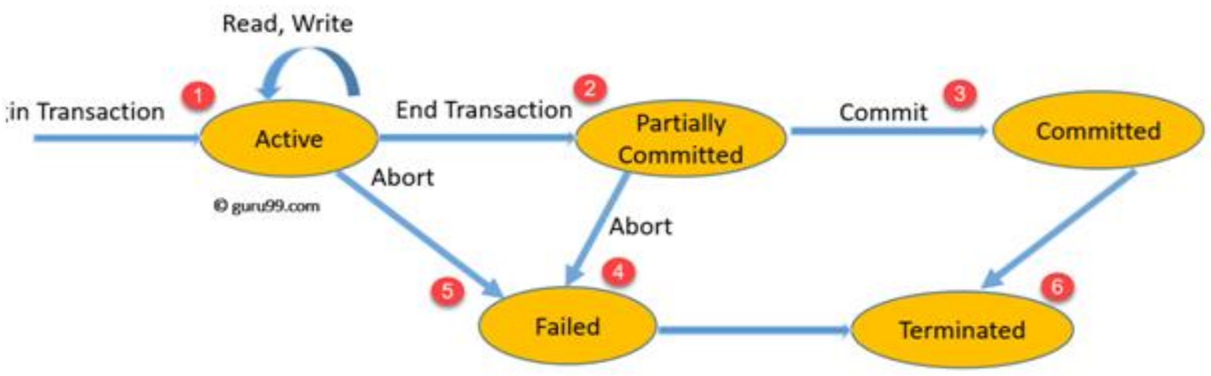
- A successful transaction can change the database from one consistent state to another
- DBMS transactions must be atomic, consistent, isolated and durable
- If the database were in an inconsistent state before a transaction, it would remain in the inconsistent state after the transaction.

## Concurrency in Transactions

A database is a shared resource accessed. It is used by many users and processes concurrently. For example, the banking system, railway, and air reservations systems, stock market monitoring, supermarket inventory, and checkouts, etc.

If concurrent access is not managed it may create issues like Hardware failure and system crashes. Concurrent execution of the same transaction, deadlock, or slow performance

## <u>State Transition Diagram</u>



## **States of Transactions**

Active State :

A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.

Partially Committed:

A transaction goes into the partially committed state after the end of a transaction.

Committed State:

When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.

Failed State :

A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.

Terminated State:

State of transaction reaches terminated state when certain transactions which are leaving the system can't be restarted.

## ACID Properties

For maintaining the integrity of data, the DBMS system you have to ensure ACID properties. ACID stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability.

- **Atomicity:**

  A transaction is a single unit of operation. You either execute it entirely or do not execute it at all. There cannot be partial execution.

- **Consistency:**

  Once the transaction is executed, it should move from one consistent state to another.

- **Isolation:**

  Transaction should be executed in isolation from other transactions . During concurrent transaction execution, intermediate transaction results from simultaneously executed transactions should not be made available to each other.

- **Durability:**

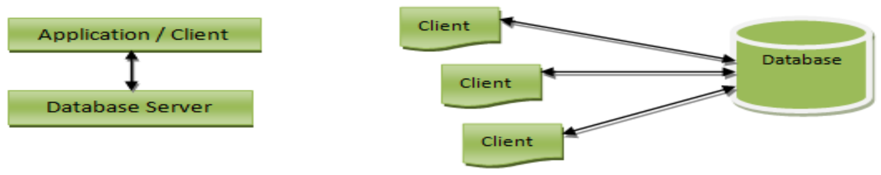
  After successful completion of a transaction, the changes in the database should persist. Even in the case of system failures.

## 11.)DATABASE ARCHITECTURE

Database architecture can be 2-tier or 3 tier architecture based on how users are connected to the database to get their request done.

They can either directly connect to the database or their request is received by intermediary layer, which synthesizes the request and then it sends to database.

### 2-tier Architecture



In 2-tier architecture, application program directly interacts with the database. There will not be any user interface or the user involved with database interaction.

In this case, the application will directly interact with the database and retreive all required data. Here no inputs from the user are required. This involves 2-tier architecture of the database.

Let us consider another example of two tier architecture.

Consider a railway ticket reservation system. How does this work? Imagine a person is reserving the ticket from Delhi to Goa on particular day. At the same time another person in some other place of Delhi is also reserving the ticket to Goa on the same day for the same train. Now there is a requirement for two tickets, but for different persons. What will reservation system do? It takes the request from both of them, and queues the requests entered by each of them. Here the request entered to application layer and request is sent to database layer. Once the request is processed in database, the result is sent back to application layer for the user.

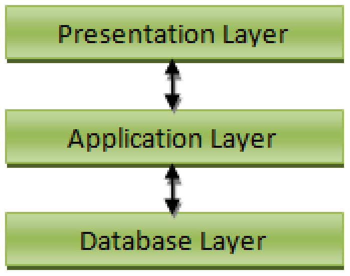
**Advantages of 2-tier Architecture**

- Easy to understand as it directly communicates with the database.
- Requested data can be retrieved very quickly, when there is less number of users.
- Easy to modify – any changes required, directly requests can be sent to database
- Easy to maintain – When there are multiple requests, it will be handled in a queue and there will not be any chaos.

**Disadvantages of 2-tier architecture**:

- It would be time consuming, when there is huge number of users. All the requests will be queued and handed one after another. Hence it will not respond to multiple users at the same time.
- This architecture would little cost effective.

## 3-tier Architecture



3-tier architecture is the most widely used database architecture. It can be viewed as below.

### Presentation layer / User layer

In this, the layer where user uses the database. He does not have any knowledge about underlying database. He simply interacts with the database as though he has all data in front of him. You can imagine this layer as a registration form where you will be inputting your details.

After pressing 'submit' button where the data goes? . You just know that your details are saved. This is the presentation layer where all the details from the user are taken, sent to the next layer for processing.

### Application layer:

In this layer ,it is the underlying program which is responsible for saving the details that you have entered, and retrieving your details to show up in the page. This layer has all the business logics like validation, calculations and manipulations of data, and then sends the requests to database to get the actual data. If this layer sees that the request is invalid, it sends back the message to presentation layer. It will not hit the database layer at all.

**<u>Data layer or Database layer</u> :**

Database layer are where actual database resides. In this layer, all the tables, their mappings and the actual data present. When you save you details from the front end, it will be inserted into the respective tables in the database layer, by using the programs in the application layer. When you want to view your details in the web browser, a request is sent to database layer by application layer. The database layer fires queries and gets the data. These data are then transferred to the browser (presentation layer) by the programs in the application layer.

**<u>Advantages of 3-tier architecture</u>**:

1. Easy to maintain and modify

Any changes requested will not affect any other data in the database. Application layer will do all the validations.

2. Improved security.

Since there is no direct access to the database, data security is increased. There is no fear of mishandling the data. Application layer filters out all the malicious actions.

3. Good performance.

Since this architecture cache the data once retrieved, there is no need to hit the database for each request. This reduces the time consumed for multiple requests and hence enables the system to respond at the same time.

**<u>Disadvantages 3-tier Architecture</u>**

It is little more complex and little more effort is required in terms of hitting the database.

## 12.)Types (or)Models of Database Management Systems

To satisfy the needs of data storage, processing and retrieval, database models of varying degrees of sophistication were devised.

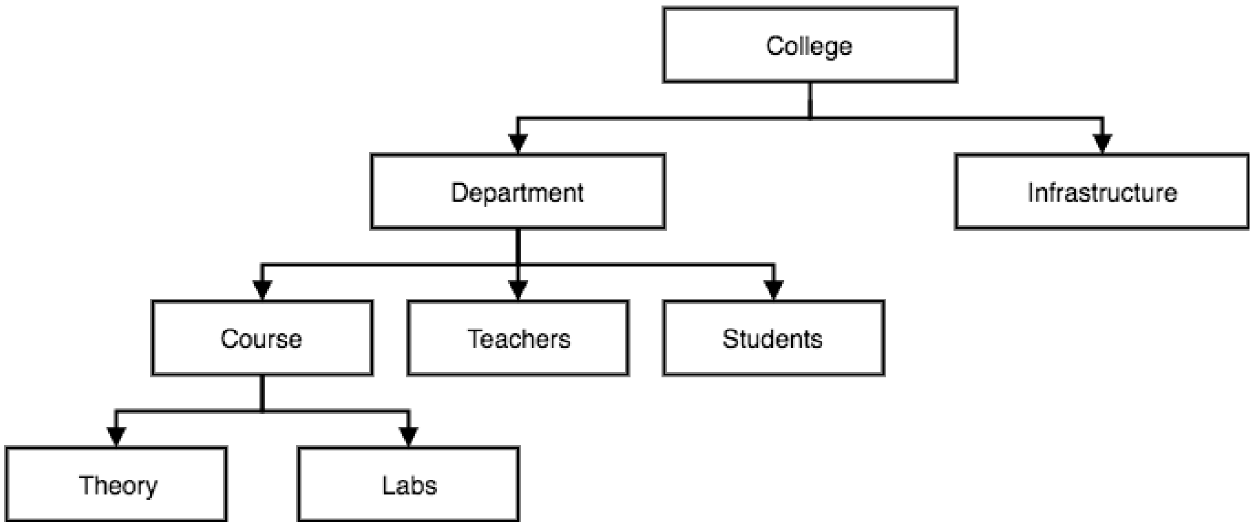
Large enterprises needed to build many independent data files containing related and different data. Data-processing activities frequently required the linking of data from several files necessitating designing data structures and database management systems that supported the automatic linkage of files.

Four database models and their corresponding management programs were developed to support the linkage of records of these types. The following database models and their management systems are in common use:

1. Hierarchical databases.
2. Network databases.
3. Relational databases.
4. Object-oriented databases

## Hierarchical Database Management System:

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. 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.



A hierarchical database is a one in which the data elements have a one-to-many relationship (1:N). The schema for a hierarchy has a single root. This kind of database model uses a tree-like structure which links a number of dissimilar elements to one primary record – the "owner" or "parent".

Each record in a hierarchical database contains information about a group of parent child relationships. The data are stored as records, each of which is a collection of fields containing only one value. The records are connected to each other through links. The structure implies that a record can have a data element repeated. Hierarchical models make the most sense where the primary focus of information gathering is on a concrete hierarchy such as a list of business departments, assets or people that will all be associated with specific higher-level primary data elements. They are very simple and fast. In the hierarchical database model the user must have some prior information about the database. Hierarchical databases were popular in early database design.

The idea behind hierarchical database models is useful for a certain type of data storage, but it is not extremely versatile. They are confined to some very specific uses.

For example, where each individual person in a company may report to a given department, the department can be used as a parent record and the individual employees will represent secondary records, each of which links back to that one parent record in a hierarchical structure.

Advantage:

Hierarchical databases relate well to anything that works through a one-to-many relationship. They can be accessed and updated rapidly because in this model, the data structure is like that of a tree, and the relationships between records are defined in advance.

These databases allow easy addition and deletion of records. These databases are good for hierarchies such as employees in an organization or an inventory of plant specimen in a museum. Data at the top of the hierarchy is accessed with great speed.

Disadvantage:

This type of database structure permits each child a relationship with only one parent, and relationships or linkages between children are not permitted, even if they make sense from a logical standpoint.

This limitation is circumvented by a repetition of data, which adds to the size of the database.

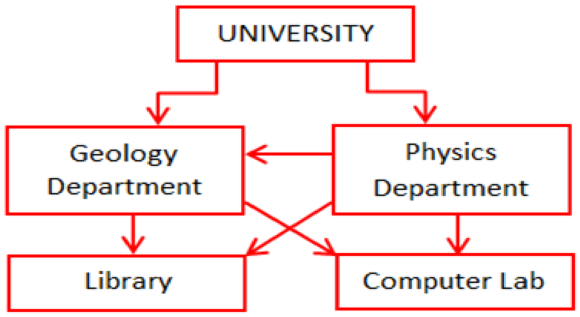
Searching for specific data requires the DBMS to run through the entire data from top to bottom until the required information is found, making queries very slow. The lower the required data in the hierarchy, the longer it takes to retrieve it.

Adding a new field or record requires the entire database to be redefined.

## Network Database Management System:

Network model is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.This was the most widely used database model.



A network database model is one in which multiple member records or files are linked to multiple owner files and vice versa. The network database model can be viewed as a net-like form where a single element can point to multiple data elements and can itself be pointed to by multiple data elements.

The network database model allows each record to have multiple parents as well as multiple child records, which can be visualized as a web-like structure of networked records.

The network model is quite similar to the hierarchical model – the hierarchical model being a subset of the network model. However, instead of using a single-parent tree hierarchy, the network model uses set theory to provide a tree-like hierarchy with the exception that child tables are allowed to have more than one parent. It supports many-to-many relationships and can be visualized as a interconnected network of records.

Advantages:

1. A Network database is conceptually simple and easy to design.
2. The data access is easier and more flexible as compared to a hierarchical model.
3. It does not allow a member to exist without a parent.
4. The main advantage of a network database is that it can handle more complex data because of its many-to-many relationship.
5. It allows for a more natural modeling of relationships between records or entities, as opposed to the hierarchical model. Because of its flexibility, it is easier to navigate and search for information in a network database.

Disadvantages:

1.The main disadvantage is that all the records in the database need to be maintained using pointers, making the whole database structure very complex.

2..The insertion, deletion and updating operations of any record require an adjustment of a large number of pointers.

3. Network databases are difficult to use by first time users.

4.Structural changes to the database are very difficult to implement.

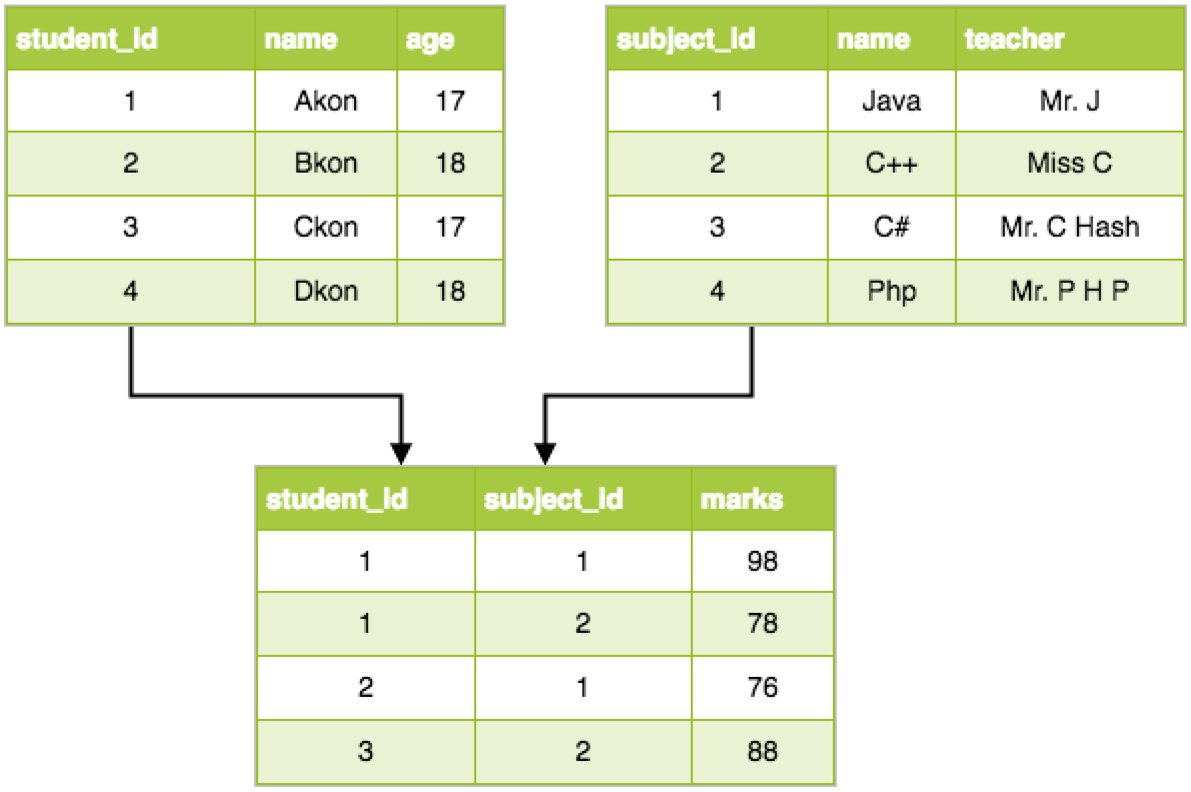
5.Difficulties are encountered while making alterations to the database because entering new data may necessitate altering the entire database.

## **Relational Databases Management System**:

A relational database is one in which data is stored in the form of tables, using rows and columns. This arrangement makes it easy to locate and access specific data within the database.

It is *“relational”*because the data within each table are related to each other. Tables may also be related to other tables. In relational databases, tables or files containing data are called *relations* (tuples), and are defined by rows (or*records*), and columns (or *attributes*) referred to as fields. Each table has a key field that mainly identifies each record (row), and on the basis of which records in different tables are related (or linked).

This kind of a relational structure makes it possible to run queries that need to retrieve data from multiple tables simultaneously. An RDBMS may also provide a visual representation of the data. For example, it may display data in a spreadsheet-like table, allowing you to view and even edit individual data elements in the table. Some RDMBS programs allow you to create forms that can streamline entering, editing, and deleting data.



| student_Id | name | age |
|---|---|---|
| 1 | Akon | 17 |
| 2 | Bkon | 18 |
| 3 | Ckon | 17 |
| 4 | Dkon | 18 |

| subject_Id | name | teacher |
|---|---|---|
| 1 | Java | Mr. J |
| 2 | C++ | Miss C |
| 3 | C# | Mr. C Hash |
| 4 | Php | Mr. P H P |

| student_Id | subject_Id | marks |
|---|---|---|
| 1 | 1 | 98 |
| 1 | 2 | 78 |
| 2 | 1 | 76 |
| 3 | 2 | 88 |

Relational DBMS software is available for large mainframe systems as well as workstations and personal computers. The need for more powerful and flexible data models to support scientific and business applications has led to extended relational data models in which table entries are no longer simple values but can be programs, text, unstructured data in the form of large binary, or in any other format which the end user needs.

Advantages:

1.Any data organized as tables consisting of rows and columns is much easier to understand.

2.Data can be stored in separate tables or files containing logically related attributes, so that huge amounts of data are segmented, making management and retrieval easier and faster.

3.Different tables from which information has to be linked and extracted can be easily managed.

4.Security and authorization control can also be implemented more easily by moving sensitive data in a given database to a separate relation with its own authorization controls.

5. Data independence is easily achieved in a relational database than in the more complicated tree or network structure.

6.Redundancy and replication of data can be minimized.

7.RDBMS offers the possibility of responding to queries by means of a language based on relational algebra and relational calculus.

8.It offers logical database independence i.e. data can be viewed in different ways by the different users.

9.Multiple users can access the database simultaneously which is not possible in other kinds of databases.

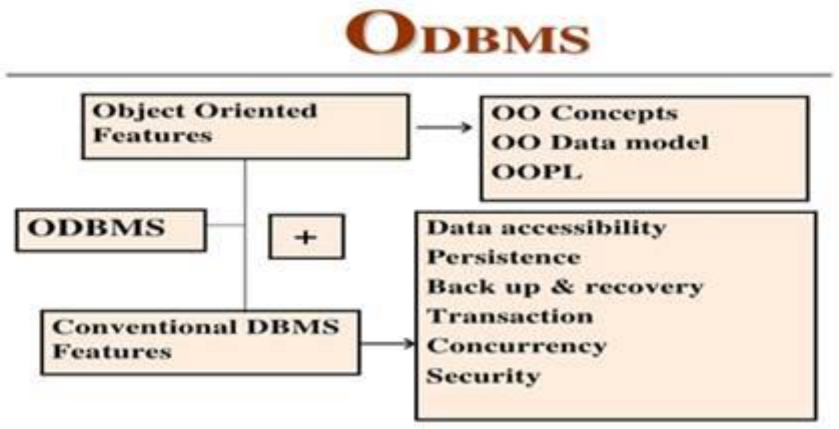
10.RDBMS also offers better backup and recovery options.

Disadvantages:

1.A major constraint, and therefore disadvantage of RDBMS is its reliance on machine performance. If the number of tables between which relationships are to be established is large, then the performance in responding to the SQL queries is affected.

2.The required hardware is complex and software expensive, increasing the overall cost of implementing RDBMS.

## Object-Oriented Database Management System:



In object-oriented databases, all data are objects. Objects may be linked to each other by an "is-part-of" relationship to represent larger, composite objects.

Classes of objects may form a hierarchy in which individual objects may inherit properties from objects higher up in the hierarchy.

Multimedia databases, in which voice, music, and video are stored along with the traditional textual information, provide a justification for viewing data as objects. Such object oriented databases are becoming increasingly important, since their structure is most flexible and adaptable. The same is true of databases of pictures, images, photographs or maps. The future of database technology is generally perceived to be an integration of the relational and object-oriented database models.

Advantages:

- Object oriented databases combine the object oriented principle with the database management principle to give a hybrid system that is more powerful than the conventional relational database management system.

- The principles of object orientation like consistency, isolation, durability, and atomicity are supported along with the principles of database systems.

- In OODBMS, working with objects in the programming language is similar to working with objects in the database. Each object is the database is identified by an object identifier called the OID which is generated by the system.

- The OODBMS is more powerful than the RDBMS if you are used to object oriented programming.

- Another advantage of using the OODBMS is that when your application requests for an object, it is sent by the database to the memory, and you work with the in-memory object.

- Any update or deletion is done to the in-memory object and these changes can later be saved to the database. This helps to avoid the frequent access to the database while updating, deleting, etc.

Disadvantages:

1.The use of object oriented database is relatively limited because we do not yet have the level of experience that we have with traditional systems.

2. The increased functionality provided by the OODBMS makes the system more complex than the traditional DBMSs.

3.object oriented database does not provide adequate security mechanisms – the database manager cannot grant access rights on individual objects or classes.

**Entity-relationship Model**

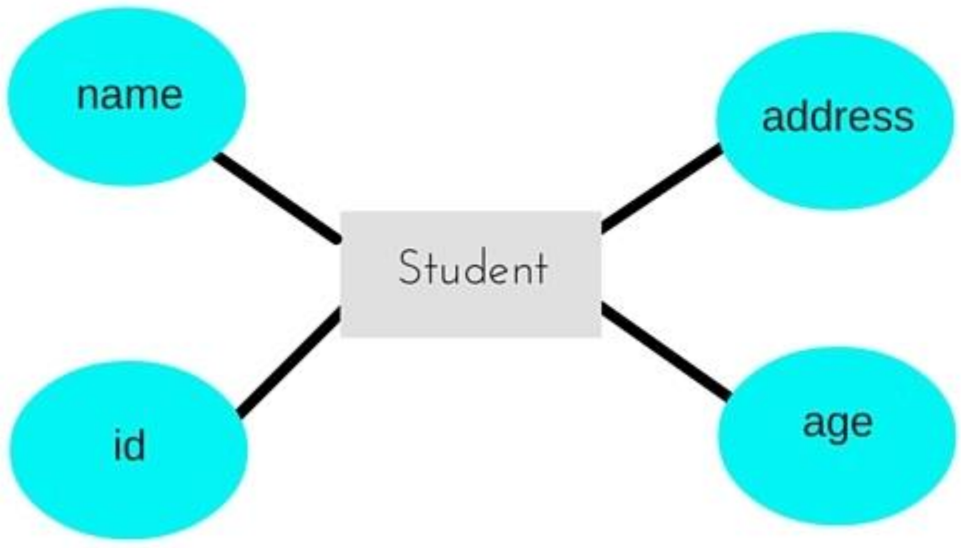
In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

Different entities are related using relationships.

E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

This model is good to design a database, which can then be turned into tables in relational model.

Let's take an example, If we have to design a School Database, then Student will be an entity with attributes name, age, address etc. As Address is generally complex, it can be another entity withattributes street name, pincode, city etc, and there will be a relationship between them.



## <u>13.)Database Users and Administrators</u>

A primary goal of a database system is to retrieve information from and store new information in the database. People who work with a database can be categorized as database users or database administrators.

### 1) Database Users and User Interfaces

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users:

**<u>Naive users</u>** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

For example, a bank teller who needs to transfer $50 from account Ato account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.
The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

**<u>Application programmers</u>** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD)tools are tools that enable an application programmer to construct forms and reports without writing a program.

**<u>Sophisticated users interact</u>** with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

**Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

**2 Database Administrators:**

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator(DBA).

The functions of a DBA include:

A database administrator's (DBA) primary job is to ensure that data is available, protected from loss and corruption, and easily accessible as needed.

**Below are some of the chief responsibilities**.

**1. Software installation and Maintenance**

A DBA often collaborates on the initial installation and configuration of a new Oracle, SQL Server etc database. The system administrator sets up hardware and deploys the operating system for the database server, then the DBA installs the database software and configures it for use. As updates and patches are required, the DBA handles this on-going maintenance.

**2. Data Extraction, Transformation, and Loading**

Known as ETL, data extraction, transformation, and loading refers to efficiently importing large volumes of data that have been extracted from multiple systems into a data warehouse environment.

This external data is cleaned up and transformed to fit the desired format so that it can be imported into a central repository.

**3. Specialised Data Handling**

Today's databases can be massive and may contain unstructured data types such as images, documents, or sound and video files. Managing a very large database (VLDB) may require higher-level skills and additional monitoring and tuning to maintain efficiency.

**4. Database Backup and Recovery**

In the case of a server failure or other form of data loss, the DBA will use existing backups to restore lost information to the system. Different types of failures may require different recovery strategies, and the DBA must be prepared for any eventuality. With technology change, it is becoming ever more typical for a DBA to backup databases to the cloud.

**5. Security**

A DBA needs to know potential weaknesses of the database software and the company's overall system and work to minimise risks.

**6. Authentication**

Setting up employee access is an important aspect of database security. DBAs control who has access and what type of access they are allowed. For instance, a user may have permission to see only certain pieces of information, or they may be denied the ability to make changes to the system.

**7. Capacity Planning**

The DBA needs to know how large the database currently is and how fast it is growing in order to make predictions about future needs. Storage refers to how much the database takes up in server and backup space. Capacity refers to usage level.

**8. Performance Monitoring**

Monitoring databases for performance issues is part of the on-going system maintenance a DBA performs. If some part of the system is slowing down processing, the DBA may need to make configuration changes to the software or add additional hardware capacity.

**9. Database Tuning**

Performance monitoring shows where the database should be tweaked to operate as efficiently as possible. The physical configuration, the way the database is indexed, and how queries are handled can all have a dramatic effect on database performance.

With effective monitoring, it is possible to proactively tune a system based on application and usage instead of waiting until a problem develops.

**10. Troubleshooting**

DBAs are on call for troubleshooting in case of any problems. Whether they need to quickly restore lost data or correct an issue to minimize damage, a DBA needs to quickly understand and respond to problems when they occur.

## Types of DBA

**1.Administrative DBA**

Work on maintaining the server and keeping it running. Concerned with backups, security, patches, replication, etc. Things that concern the actual server software.

**2.Development DBA**

works on building queries, stored procedures, etc. that meet business needs. This is the equivalent of the programmer. You primarily write T-SQL.

**3.Architect**

Design schemas. Build tables, FKs, PKs, etc. Work to build a structure that meets the business needs in general. The design is then used by developers and development DBAs to implement the actual application.

**4.Data Warehouse DBA**

Newer role, but responsible for merging data from multiple sources into a data warehouse. May have to design warehouse, but cleans, standardizes, and scrubs data before loading. In SQL Server, this DBA would use DTS heavily.

**5.OLAP DBA**

Builds multi-dimensional cubes for decision support or OLAP systems. The primary language in SQL Server is MDX, not SQL here.

**6.Application DBA**

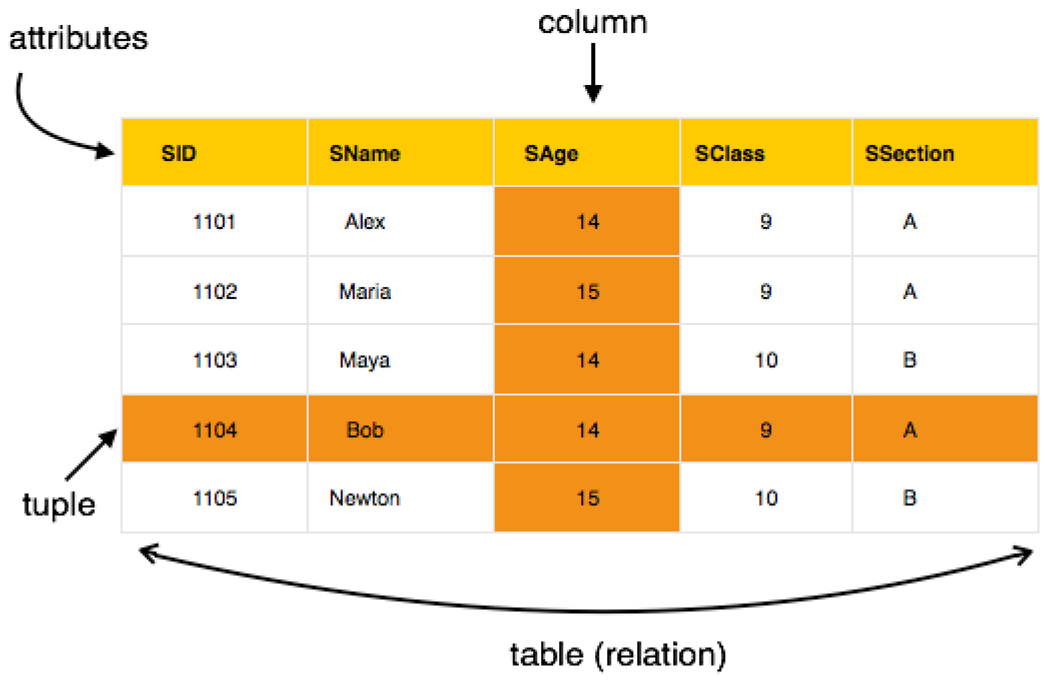
Application DBAs straddle the fence between the DBMS and the application software and are responsible for ensuring that the application is fully optimized for the database and vice versa. They usually manage all the application components that interact with the database and carry out activities such as application installation and patching, application upgrades, database cloning, building and running data cleanup routines, data load process management, etc.

# UNIT-2

# RELATIONAL ALGEBRA & CALCULUS

## 1. STRUCTURE OF RELATIONAL MODEL

Relational data model is the primary data model, which is used widely for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.



| SID | SName | SAge | SClass | SSection |
|---|---|---|---|---|
| 1101 | Alex | 14 | 9 | A |
| 1102 | Maria | 15 | 9 | A |
| 1103 | Maya | 14 | 10 | B |
| 1104 | Bob | 14 | 9 | A |
| 1105 | Newton | 15 | 10 | B |

In Relational model has the following :

**Tables**

In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represent the attributes.

**Tuple**

A single row of a table, which contains a single record for that relation is called a tuple.

**Relation instance**

A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

**Relation schema**

A relation schema describes the relation name (table name), attributes, and their names.

**Relation key**

Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

**Attribute domain**

Every attribute has some pre-defined value scope, known as attribute domain.

## Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

***Key Constraints***

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subset, these are called ***candidate keys***.

Key constraints force that −

- In a relation with a key attribute, no two tuples can have identical values for key attributes.
- A key attribute cannot have NULL values.

Key constraints are also referred to as Entity Constraints.

***Domain Constraints***

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

***Referential integrity Constraints***

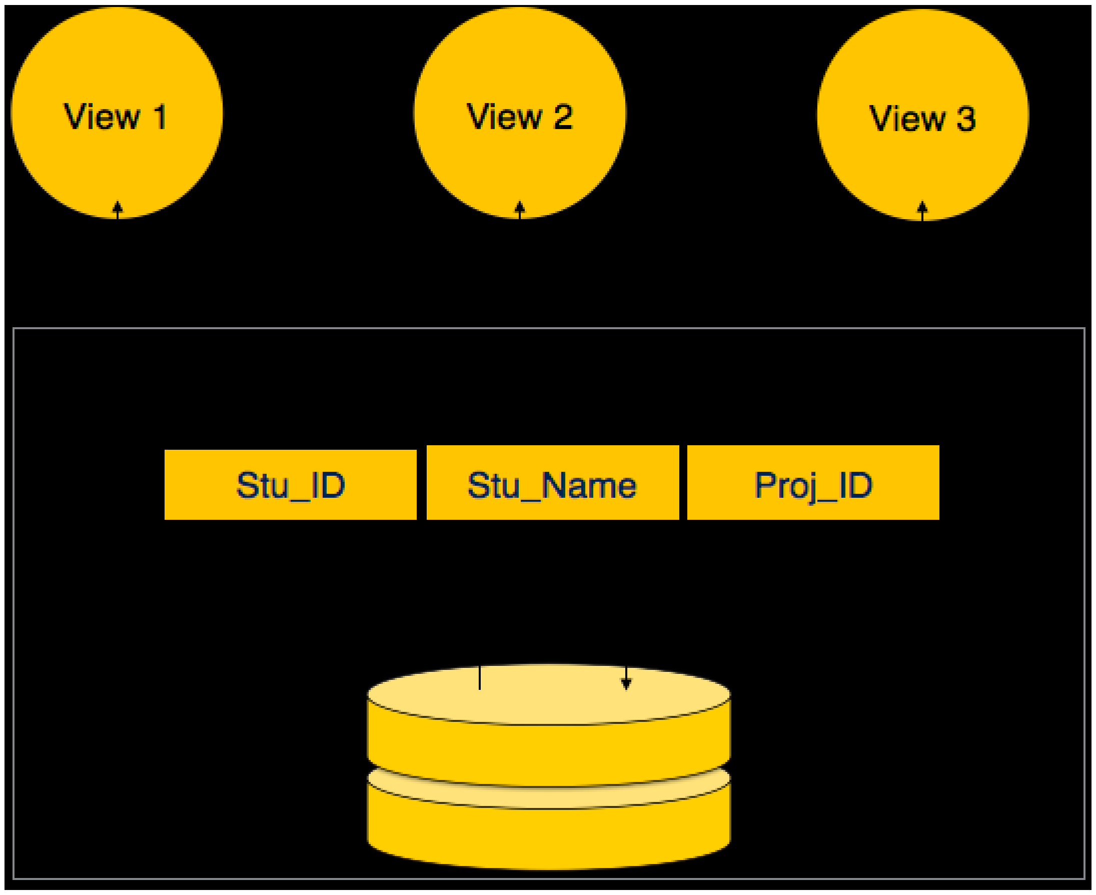
Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

## **Database Schema**

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



A database schema can be divided broadly into two categories −

- **Physical Database Schema** –

  This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.

- **Logical Database Schema** :

  It defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

***Database Instance***

A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time.

A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

## Relational Algebra in DBMS

Relational algebra is a **procedural** query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data.

### Types of operations in relational algebra

We have divided these operations in two categories:

1. Basic Operations
2. Derived Operations

### Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

### Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

If you understand little bit of SQL then you can think of it as a where clause in SQL, which is used for the same purpose.

**Syntax of Select Operator (σ)**

```
σ Condition/Predicate(Relation/Table name)
```

**Select Operator (σ) Example**

```
Table: CUSTOMER
---------------


Customer_Id     Customer_Name      Customer_City
-----------           -------------                    -------------
C10100          ARUN                          Agra
C10111          RAJESH                       Agra
C10115          CHARLES                    Noida
C10117          AJIT                           Delhi
C10118          DAVID                        Delhi
```

**Query:**

```
σ Customer_City="Agra" (CUSTOMER)
```

**Output:**

```
Customer_Id   Customer_Name    Customer_City
-----------   -------------    -------------
C10100       ARUN            Agra
C10111       RAJESH          Agra
```

**Project Operator (∏)**

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the Select statement in SQL.

**Syntax**

```
∏ column_name1, column_name2, ...., column_nameN(table_name)
```

**Example**

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator ∏.

```
Table: CUSTOMER

Customer_Id    Customer_Name    Customer_City
-----------    -------------    -------------
C10100         ARUN             Agra
C10111         RAJESH           Agra
C10115         CHARLES          Noida
C10117         AJIT             Delhi
C10118         KUMAR            Delhi
```

**Query:**

```
∏ Customer_Name, Customer_City (CUSTOMER)
```

**Output:**

```
Customer_Name            Customer_City
-------------              -------------
ARUNS                        Agra
RAJESH                       Agra
CHARLES                     Noida
AJIT                        Delhi
KUMAR                        Delhi
```

**Union Operator (∪)**

Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

we have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

The rows (tuples) that are present in both the tables will only appear once in the union set. There are no duplicates present after the union operation.

**Syntax :**

```
table_name1 ∪ table_name2
```

**Example:**

Table 1: COURSE

| Course_Id | Student_Name | Student_Id |
|---|---|---|
| C101 | ARUN | S901 |
| C104 | ARUN | S901 |
| C106 | RAJESH | S911 |
| C109 | KUMAR | S921 |
| C115 | DAVID | S931 |

Table 2: STUDENT

| Student_Id | Student_Name | Student_Age |
|---|---|---|
| S901 | ARUN | 19 |
| S911 | RAJESH | 18 |
| S921 | KUMAR | 19 |
| S931 | DAVID | 17 |
| S941 | CHARLES | 16 |
| S951 | RAMESH | 18 |

**Query:**

∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)

**Output:**

| Student_Name |
|---|
| ARUN |
| CHARLES |
| KUMAR |
| DAVID |
| RAMES |
| RAJESH |

**Intersection Operator (∩)**

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

**Syntax :**

```
table_name1 ∩ table_name2
```

**Example**

Table 1: COURSE

| Course_Id | Student_Name | Student_Id |
|---|---|---|
| C101 | ARUN | S901 |
| C104 | ARUN | S901 |
| C106 | CHARLES | S911 |
| C109 | DAVID | S921 |
| C115 | RAJESH | S931 |

Table 2: STUDENT

| Student_Id | Student_Name | Student_Age |
|---|---|---|
| S901 | ARUN | 19 |
| S911 | CHARLES | 18 |
| S921 | DAVID | 19 |
| S931 | KUMAR | 17 |
| S941 | AJIT | 16 |
| S951 | MOHAN | 18 |

**Query:**

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

**Output:**

```
Student_Name
------------
ARUN
CHARLES
DAVID
KUMAR
```

**Set Difference (-)**

Set Difference is denoted by – symbol. Lets say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference R1 – R2.

**Syntax of Set Difference (-)**

```
table_name1 - table_name2
```

**Example**

Lets take the same tables COURSE and STUDENT that we have seen above.

**Query:**

Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

```
∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)
```

**Output:**

```
Student_Name
------------
CHARLES
RAJESH
```

**Cartesian product (X)**

Cartesian Product is denoted by X symbol. we have two relations R1 and R2 then the cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2.

**Syntax:**

```
R1 X R2
```

**Example:**

Table 1: R

```
Col_A    Col_B
-----      ------
AA        100
BB        200
CC        300
```

Table 2: S

```
Col_X      Col_Y
-----          -----
XX            99
YY            11
ZZ           101
```

**Query:**

Cartesian product of table R and S.

```
R X S
```

**Output:**

```
Col_A    Col_B     Col_X     Col_Y
-----       ------     ------          ------
AA       100          XX        99
AA       100         YY        11
AA       100         ZZ        101
BB       200         XX        99
BB       200         YY        11
BB       200         ZZ        101
CC       300         XX        99
CC       300         YY        11
CC       300          ZZ        101
```

The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has 3×3 = 9 rows.

## Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

**Rename (ρ) Syntax:**

ρ(new_relation_name, old_relation_name)

### Example

Lets  have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

| Customer_Id | Customer_Name | Customer_City |
|---|---|---|
| C10100 | ARUN | Agra |
| C10111 | DAVID | Agra |
| C10115 | CHARLES | Noida |
| C10117 | AJIT | Delhi |
| C10118 | SURESH | Delhi |

**Query:**

ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER))

**Output:**

| CUST_NAMES |
|---|
| ARUN |
| DAVID |
| CHARLES |
| AJIT |
| SURESH |

## Relational Calculus

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn’t tell how to retrieve it.

### Types of Relational Calculus

### 1. Tuple Relational Calculus (TRC)

Tuple relational calculus is used for selecting those tuples that satisfy the given condition.
Table: Student

```
First_Name    Last_Name    Age
----------    ---------    ----
ARUN          KUMAR        30
BALA          KUMAR        31
Raj           KUMAR        27
SANTHOSH      Raj          28
```

Lets write relational calculus queries.

Query to display the last name of those students where age is greater than 30

```
{ t.Last_Name | Student(t) AND t.age > 30 }
```

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.

The result of the above query would be:

```
Last_Name
---------
KUMAR
```

Query to display all the details of students where Last name is ‘KUMAR’

```
{ t | Student(t) AND t.Last_Name = 'KUMAR' }
```

**Output:**

```
First_Name    Last_Name    Age
----------    ---------    ----
ARUN           KUMAR        30
BALA           KUMAR        31
RAJ            KUMAR        27
```

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

In domain relational calculus the records are filtered based on the domains.

Table: Student

```
First_Name    Last_Name    Age
----------    ---------    ----
ARUN          KUMAR         30
BALA          KUMAR         31
Raj           KUMAR         27
Santhosh      RAJ           28
```

Query to find the first name and age of students where student age is greater than 27

```
{< First_Name, Age > | ∈ Student ∧ Age > 27}
```

**Note:**
The symbols used for logical operators are: ∧ for AND, ∨ for OR and ┐ for NOT.

**Output:**

```
First_Name     Age
----------      ----
ARUN           30
BALA           31
santhosh       28
```

## **Extended Operators in Relational Algebra**

In Relational Algebra, Extended Operators are those operators that are derived from the basic operators. There are mainly three types of Extended Operators, namely:

1.Intersection

2.Divide

3.Join

Let us consider two tables named as A and B.

A –

| RollNo | Name | Marks |
|---|---|---|
| 1 | ARUN | 98 |
| 3 | BALA | 79 |
| 4 | DAVID | 88 |

B –

| RollNo | Name | Marks |
|---|---|---|
| 1 | ARUN | 98 |
| 2 | CHARLES | 87 |
| 3 | BALA | 79 |
| 4 | DAVID | 88 |

1) Intersection

Intersection works on the relation as 'this and that'. In relational algebra, A ∩ B returns a relation instance that contains every tuple that occurs in relation to instance A and relation instance B (both together). Here, A and B need to be union-compatible, and the schema of both result and A must be identical.

Syntax:

```
SELECT * FROM A INTERSECT SELECT * FROM B;
```

| Roll No | Name | Marks |
|---|---|---|
| 1 | ARUN | 98 |
| 3 | BALA | 79 |
| 4 | DAVID | 88 |

2) Divide

Divide operator is used for the queries that contain the keyword ALL.

For e.g. – Find all the students who has chosen additional subjects C and JAVA.

Student –

| Student Name | Subject |
|---|---|
| ARUN | C |
| ARUN | JAVA |
| DAVID | Network Security |
| DAVID | Data Mining |
| KUMAR | Network Security |
| KUMAR | DATABASE |
| CHARLES | C |
| CHARLES | JAVA |

Subject –

| Subject Name |
|---|
| C |
| JAVA |

Output: Student ÷ Subject

| Student |
|---|
| ARUN |
| CHARLES |

# UNIT -3

# SQL

## 1. SQL

Structured Query Language (SQL) is the standard language for data manipulation in a DBMS.

Following are types of SQL Statements

1. Data Definition Language (DDL) allows you to create objects like Schemas, Tables in the database
2. Data Control Language (DCL) allows you to manipulate and manage access rights on database objects
3. Data Manipulation Language (DML) is used for searching, inserting, updating, and deleting data, which will be partially covered in this programming tutorial.

**Why SQL?**

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

## Applications of SQL

SQL is one of the most widely used queries language over the databases.SQL has the following applications:

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

## Advantages of SQL

There are the following advantages of SQL:

### High speed

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

### No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

## Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

## Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

## Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

## Multiple data view

Using the SQL language, the users can make different views of the database structure.

## 2.SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

## SQL languages & its types:

### 1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

**a. CREATE** It is used to create a new table in the database.

**Syntax:**

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
```

**b. DROP:**

It is used to delete both the structure and record stored in the table.

**Syntax**

DROP TABLE ;

**c. ALTER:**

It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or to add a new attribute.

**Syntax:**

To add a new column in the table

```
ALTER TABLE table_name ADD column_name COLUMN-definition;
```

To modify existing column in the table:

```
ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

**d. TRUNCATE:**

It is used to delete all the rows from the table and free the space containing the table.

**Syntax:**

```
TRUNCATE TABLE table_name;
```

## 2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

**a. INSERT:**

The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:**

INSERT INTO TABLE_NAME(col1, col2, col3,.... col N)  VALUES (value1, value2, value3, .... valueN);

Or

INSERT INTO TABLE_NAME VALUES (value1, value2, value3, .... valueN);

**UPDATE:**

This command is used to update or modify the value of a column in the table.

**Syntax:**

UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

**DELETE:**

It is used to remove one or more row from a table.

**Syntax:**

DELETE FROM table_name [WHERE condition];

## 3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

1.Grant

2.Revoke

**a. Grant:**

It is used to give user access privileges to a database.

**b.Revoke:**

It is used to take back permissions from the user.

## 4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

**a. Commit:**

Commit command is used to save all the transactions to the database.

**Syntax:**

COMMIT;

**b. Rollback:**

Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:** ROLLBACK;

**c. SAVEPOINT:**

It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

SAVEPOINT SAVEPOINT_NAME;

## 5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

SELECT

a. **SELECT:**

This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

**Syntax:**

SELECT expressions  FROM TABLES  WHERE conditions;

## 3.SQL  SELECT  Statement

Select statement is used to fetch data from relational database. A relational database is organized collection of data.

SQL select statement or SQL select query is used to fetch data from one or more than one tables. We can fetch few columns, few rows or entire table using SELECT Query based on the requirement.

Example: we have a table EMPLOYEES with the following data:

```
+------+----------+---------+----------+
| SSN  | EMP_NAME | EMP_AGE |EMP_SALARY|
+------+----------+---------+----------+
| 101 |  DINESH    | 23       |  9000.00 |
| 102 |  ARUN      | 24       |  2550.00 |
| 103 | RAMESH     | 19       |  2444.00 |
| 104 | VIMAL      | 29       |  6588.00 |
| 105 | CHARLES    | 21       |  1400.00 |
| 106 | RAM        | 24       |  8900.00 |
| 107 | DAVID      | 20       | 18300.00 |
+------+----------+---------+----------+
```

**SELECT Syntax**

**Select Statement Example – Retrieve single column**

**Syntax:**

```
SELECT column_name FROM table_name;


Query:
```

```
SELECT EMP_NAME FROM EMPLOYEES;
```

Here column_name is the name of the column for which we need to fetch data and table_name is the name of the table in the database.

**Output of the above Query:**

```
+----------+
| EMP_NAME |
+------+---+
| DINESH    |
| ARUN       |
| RAMESH   |
| VIMAL      |
| CHARLES  |
| RAM          |
| DAVID       |
+----------+
```

**More than one columns:**

**Syntax:**

```
SELECT column_name_1, column_name_2, ... FROM table_name;
```

To fetch multiple columns SSN & EMP_NAME from the above table EMPLOYEES, we can write the SELECT Query like this:

Query:

```
SELECT SSN, EMP_NAME FROM EMPLOYEES;
```

**Output of the above query:**

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

| SSN  | EMP_NAME |
+------+----------+
|  101 |    DINESH   |
|  102 |     ARUN     |
|  103 |   RAMESH   |
|  104 |     VIMAL    |
|  105 |  CHARLES  |
|  106 |      RAM      |
|  107 |     DAVID    |
+------+----------+
```

**For fetching entire table:**

```
SELECT * FROM table_name;
```

Query:

```
SELECT * FROM EMPLOYEES;
```

**Result:**

```
+------+----------+---------+----------+
|SSN   | EMP_NAME  | EMP_AGE |EMP_SALARY|
+------+----------+---------+----------+
|  101 |     DINESH    |  23            |  9000.00 |
|  102 |       ARUN      |  24            |  2550.00 |
|  103 |     RAMESH  |  19            |  2444.00 |
|  104 |       VIMAL    |  29            |  6588.00 |
|  105 |    CHARLES |  21            |  1400.00 |
|  106 |         RAM       |  24            |  8900.00 |
|  107 |        DAVID    |  20            | 18300.00 |
+------+----------+---------+----------+
```

## 4.SQL USING SELECT CLAUSE

### SELECT statement Clause

**If we want to fetch only few rows or want the rows in an particular order or grouped rows etc.**

These clauses are optional but they are frequently used in SQL statements in the real world database operations.

**WHERE Clause in SQL:**

WHERE clause filters the records, with the help of this clause we specify a condition in the SQL statement and only those records that fulfill the condition are retrieved from database.

**ORDER BY Clause in SQL:**

ORDER BY clause is used to return the rows in ascending or descending order of the data.

**GROUP BY Clause in SQL:**

GROUP BY clause groups the rows with same data, this is often used along with the aggregate functions.

**HAVING Clause in SQL:**

HAVING clause filters the records, just like WHERE clause, however it is used along with the GROUP BY clause so it filters the records from the output set of rows produced by GROUP BY clause

## 5.SQL WHERE Clause

**Where clause** is used to fetch a particular row or set of rows from a table. This clause filters records based on given conditions and only those row(s) comes out as result that satisfies the condition defined in WHERE clause of the SQL query.

### SQL Where Clause Syntax

```
SELECT Column_name1, Column_name2, ....
FROM Table_name
WHERE Condition;
```

Here we have used the where clause with the SQL SELECT statement, however we can use this clause with other SQL statements as well such as UPDATE, DELETE etc.

### Example – SQL WHERE Clause

We have a table EMPLOYEES:

```
+------+----------+---------+----------+
|SSN   | EMP_NAME | EMP_AGE |EMP_SALARY|
+------+----------+---------+----------+
|  101 | DINESH   |  23     |  9000.00 |
|  102 | ARUN     |  24     |  2550.00 |
|  103 | RAMESH   |  19     |  2444.00 |
|  104 | VIMAL    |  29     |  6588.00 |
|  105 | CHARLES  |  21     |  1400.00 |
|  106 | RAM      |  24     |  8900.00 |
|  107 | DAVID    |  20     | 18300.00 |
+------+----------+---------+----------+
```

To fetch the name of the employees who are more than 23 years old. The SQL statement would look like this –

**Query:**

**SELECT EMP_NAME FROM EMPLOYEES WHERE EMP_AGE > 23;**

**Result:**

```
+----------+
| EMP_NAME |
+----------+
|   ARUN    |
|  VIMAL    |
|    RAM      |
+----------+
```

Fetch all the details of employees having salary greater than 6000.

**Query:**

**SELECT * FROM EMPLOYEES WHERE EMP_SALARY > 6000;**

**Result:**

```
+------+----------+---------+----------+
|SSN   | EMP_NAME | EMP_AGE |EMP_SALARY|
+------+----------+---------+----------+
|  101 | DINESH            |  23           |  9000.00 |
|  104 | VIMAL              |  29           |  6588.00 |
|  106 | RAM                 |  24           |  8900.00 |
|  107 | DAVID              |  20          | 18300.00 |
+------+----------+---------+----------+
```

## SQL where clause with multiple conditions

Lets take the same table:

```
+------+----------+---------+----------+
|SSN   | EMP_NAME | EMP_AGE |EMP_SALARY|
+------+----------+---------+----------+
|  101 | DINESH       |  23        |  9000.00 |
|  102 | ARUN         |  24        |  2550.00 |
|  103 | RAMESH       |  19        |  2444.00 |
|  104 | VIMAL        |  29        |  6588.00 |
|  105 | CHARLES      |  21        |  1400.00 |
|  106 | RAM          |  24        |  8900.00 |
|  107 | DAVID        |  20        | 18300.00 |
+------+----------+---------+----------+
```

Lets fetch the employee details where employee age is greater than 23 and salary is greater than 5000. For such query we have to use multiple conditions in where clause.

**Query:**

**SELECT * FROM EMPLOYEES WHERE EMP_SALARY > 5000 AND EMP_AGE > 23;**

**Result:**

```
+------+----------+---------+----------+
|SSN   | EMP_NAME | EMP_AGE |EMP_SALARY|
+------+----------+---------+----------+
|  104 | VIMAL        |  29        |  6588.00 |
|  106 | RAM          |  24        |  8900.00 |
+------+----------+---------+----------+
```

**Another multiple conditions example:**

Fetch the employee names, where either employee age is less than 20 or salary is less than 5000.

**Query:**

```
SELECT EMP_NAME

FROM EMPLOYEES

WHERE EMP_SALARY < 5000 OR EMP_AGE < 20;
```

**Result:**

```
+----------+
| EMP_NAME |
+----------+
| RAMESH   |
| ARUN     |
| CHARLES  |
+----------+
```

## 6.SQL AND, OR and NOT Operators

### SQL AND Operator

When multiple conditions are joined using AND operator, only those rows will be fetched from the database which meets all the conditions.

### AND Operator Syntax

```
SELECT column_name1, column_name2, ...
FROM table_name
WHERE condition_1 AND condition_2 ...;
```

**Example**

Table: STUDENT

```
+----+--------------+-----+-----------+----------+
| ID | STU_NAME     | AGE | ADDRESS   | BRANCH   |
+----+--------------+-----+-----------+----------+
| 11 | ARUN         |  22 | Agra      |  ECE     |
| 12 | BALA         |  23 | Delhi     |  CSE     |
| 13 | MAHESHe      |  23 | Gurgaon   |  CL      |
| 14 | CHARLES      |  24 | Noida     |  ME      |
| 15 | DAVID        |  26 | Delhi     |  EE      |
+----+--------------+-----+-----------+----------+
```

SQL statements to fetch the name of the students where student age is “greater than 23” and address is “Delhi”

**Query:**

```
SELECT STU_NAME
FROM STUDENT
WHERE AGE > 23 AND ADDRESS = 'Delhi';
```

**Result:**

```
+----------+
| STU_NAME |
+----------+
| DAVID    |
+----------+
```

### SQL OR Operator

When multiple conditions are joined using OR operator, all those rows will be fetched from the database which meet any of the given conditions.

### OR Operator Syntax

```
SELECT column_name1, column_name2, ...
FROM table_name
WHERE condition_1 OR condition_2 ...;
```

### SQL OR Example

Table: EMPLOYEE

```
+----+--------------+-----+-----------+----------+
| ID | EMP_NAME     | AGE | ADDRESS   | SALARY   |
+----+--------------+-----+-----------+----------+
| 90 | David        |  30 | Agra      |  10000   |
| 91 | DINESH       |  31 | Delhi     |   9000   |
| 92 | DAVID        |  29 | Gurgaon   |  11000   |
| 93 | RAHUL        |  33 | Noida     |  19000   |
| 94 | PRAKASH      |  35 | Agra      |   9900   |
+----+--------------+-----+-----------+----------+
```

SQL statements to fetch the details of the employees, where either employee age is “greater than 30” or address is “Agra”.

**Query:**

```
SELECT *
FROM EMPLOYEE
WHERE AGE > 30 OR ADDRESS = 'Agra';
```

**Result:**

```
+----+--------------+-----+-----------+----------+
| ID | EMP_NAME     | AGE | ADDRESS   | SALARY   |
+----+--------------+-----+-----------+----------+
| 90 | David        |  30 | Agra      |  10000   |
| 91 | DINESH       |  31 | Delhi     |   9000   |
| 93 | RAHUL        |  33 | Noida     |  19000   |
| 94 | PRAKASH      |  35 | Delhi     |   9900   |
+----+--------------+-----+-----------+----------+
```

## SQL NOT Operator

When a NOT operator is used with a condition, only those rows will fetched from database **which do not meet** the given condition.

### NOT Operator Syntax

```
SELECT column_name1, column_name2, ...
FROM table_name
WHERE NOT condition;
```

### SQL NOT Example

**Table: ORDER**

```
+----+--------------+-----+-----------+----------+
| ID | CUSTOMER_NAME| AGE | ADDRESS   | AMOUNT   |
+----+--------------+-----+-----------+----------+
| 70 | RAM          |  30 | Sector121 |  11000   |
| 71 | KUMAR        | 31  | Sector122 |    900   |
| 72 | SATHIS       |  29 | Sector62  |   1000   |
| 73 | DAVID        |  33 | Sector61  |   1000   |
| 74 | MAHESH       |  35 | Sector121 |    500   |
+----+--------------+-----+-----------+----------+
```

The following SQL statement selects all the order id and customer name where address is NOT Sector121.

**Query:**

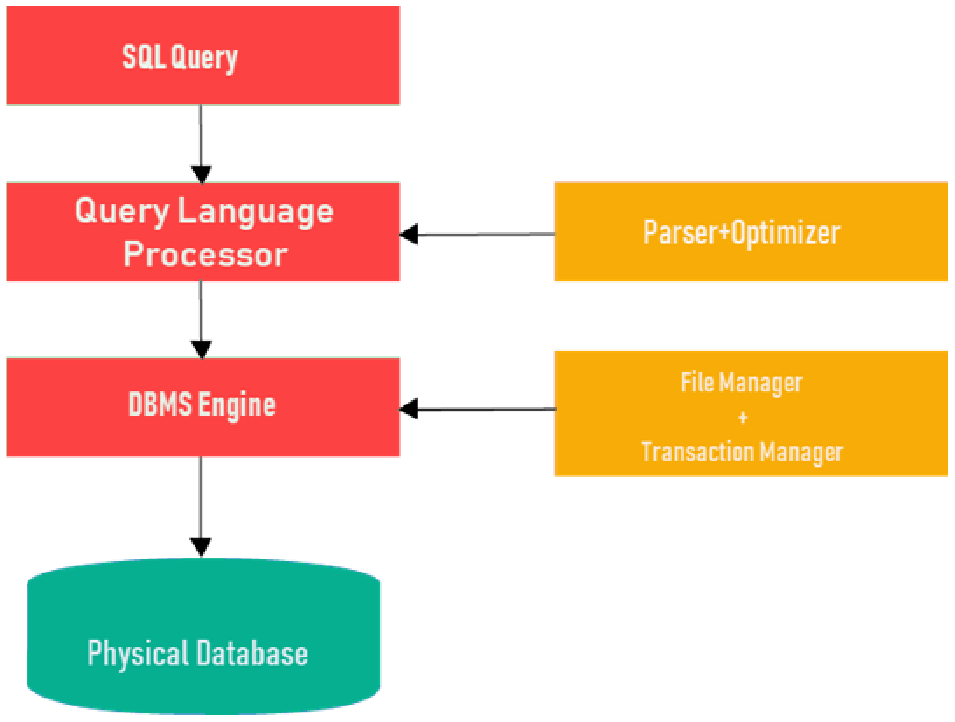
```
SELECT ID, CUSTOMER_NAME
FROM ORDER
WHERE NOT ADDRESS = 'Sector121';
```

**Result:**

```
+----+--------------+
| ID | CUSTOMER_NAME|
+----+--------------+
| 71 |      KUMAR         |
| 72 |       SATHIS       |
| 73 |       DAVID        |
+----+--------------+
```

## 7. SQL PROCESS/SQL STRUCTURE

When you want to execute an SQL command for any DBMS system, you need to find the best method to carry out your request, and SQL engine determines how to interpret that specific task.



### Query processor

A query processor is one of the major components of a relational database or an electronic database in which data is stored in tables of rows and columns.

A user, or an applications program, interacts with the query processor and the query processor, in turn interacts with the storage engine. The query processor receives an instruction or instructions written in Structured Query Language (SQL), chooses a plan for executing the instructions and carries out the plan.

Query processors come with the following components,

DDL Interpreter:

It interprets the DDL statements and records the definitions in data dictionary.

DML Compiler:

It translates the DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation understands.

Optimization

The SQL syntax is transformed into a series of operations that can be performed on data and its indices. The raw query plan, as it is known, is optimized to make it more efficient before it is executed.

*Database Engine*

A database engine is composed of the component of the system that actually stores and retrieves data.

Physical Database:

Physical database design translates the logical data model into a set of SQL statements that define the database. For relational database systems, it is relatively easy to translate from a logical data model into a physical database. Rules for translation: Entities become tables in the physical database.

**In the above diagram,**

- The first step is to transform the query into a standard form.
- A query is translated into SQL and into a relational algebraic expression. During this process, Parser checks the syntax and verifies the relations and the attributes which are used in the query.
- The second step is Query Optimizer. In this, it transforms the query into equivalent expressions that are more efficient to execute.
- The third step is Query evaluation. It executes the above query execution plan and returns the result.

## 8.SQL Set Operation

The SQL Set operation is used to combine the two or more SQL SELECT statements.

### Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus

### Union

- ➢ The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- ➢ In the union operation, all the number of data type and columns must be same in both the tables on which UNION operation is being applied.
- ➢ The union operation eliminates the duplicate rows from its result set.

**Syntax**

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

**Example:**

**First**

| ID | NAME |
|---|---|
| 1 | ARUN |
| 2 | BALA |
| 3 | DINESH |

**Second**

| ID | NAME |
|---|---|
| 3 | DINESH |
| 4 | RAMESH |
| 5 | David |

Union SQL query will be:

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The result set table will look like:

| ID | NAME |
|---|---|
| 1 | ARUN |
| 2 | BALA |
| 3 | DINESH |
| 4 | RAMESH |
| 5 | David |

## 2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

**Syntax:**

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

**Query:**

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The result set table will look like:

| ID | NAME |
|---|---|
| 1 | ARUN |
| 2 | BALA |
| 3 | DINESH |
| 3 | DINESH |
| 4 | RAMESH |
| 5 | David |

## 3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

**Syntax**

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

**Example:**

**Using the above First and Second table.**

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The result set table will look like:

| ID | NAME |
|---|---|
| 3 | DINESH |

## 4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

**Syntax:**

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

**Example**

**Using the above First and Second table.**

Minus query will be:

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

The result set table will look like:

| **ID** | **NAME** |
|---|---|
| 1 | ARUN |
| 2 | BALA |

9. **SQL Aggregate Functions**

SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.

It is also used to summarize the data.

Types of SQL Aggregation Function

1.Count( )

2.Sum( )

3.Avg(  )

4.Max( )

5.Min( )

## 1. ***COUNT FUNCTION***

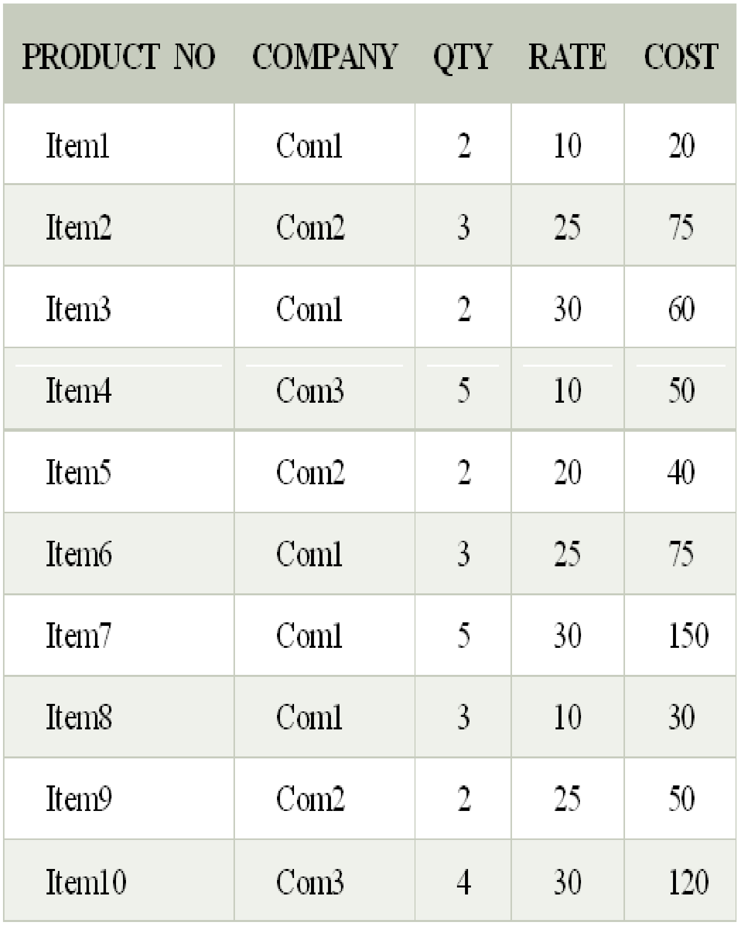
COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax:

COUNT(*) or COUNT( [ALL|DISTINCT] expression )

PRODUCT_MAST



| PRODUCT NO | COMPANY | QTY | RATE | COST |
|---|---|---|---|---|
| Item1 | Com1 | 2 | 10 | 20 |
| Item2 | Com2 | 3 | 25 | 75 |
| Item3 | Com1 | 2 | 30 | 60 |
| Item4 | Com3 | 5 | 10 | 50 |
| Item5 | Com2 | 2 | 20 | 40 |
| Item6 | Com1 | 3 | 25 | 75 |
| Item7 | Com1 | 5 | 30 | 150 |
| Item8 | Com1 | 3 | 10 | 30 |
| Item9 | Com2 | 2 | 25 | 50 |
| Item10 | Com3 | 4 | 30 | 120 |

**COUNT( )**

```
SELECT COUNT(*)  FROM PRODUCT_MAST;
```

Output:

```
10
```

**COUNT with WHERE**

```
SELECT COUNT(*)  FROM PRODUCT_MAST WHERE RATE>=20;
```

Output:

```
7
```

**COUNT( ) with DISTINCT**

```
SELECT COUNT(DISTINCT COMPANY) FROM PRODUCT_MAST;
```

Output:

```
3
```

**COUNT( ) with GROUP BY**

```
SELECT COMPANY COUNT(*) FROM PRODUCT_MAST GROUP BY
COMPANY;
```

Output:

```
Com1    5

Com2    3

Com3    2
```

**COUNT( ) with HAVING**

SELECT COMPANY COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY  HAVING COUNT(*)>2;

Output:

```
Com1    5

Com2    3
```

2. **SUM Function**

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

SUM( )  or  SUM( [ALL|DISTINCT] expression )

**SUM(  )**

SELECT SUM(COST) FROM PRODUCT_MAST;

Output:

```
670
```

**SUM( ) with WHERE**

SELECT SUM(COST)  FROM PRODUCT_MAST  WHERE QTY>3;

Output:

```
320
```

**SUM( ) with GROUP BY**

SELECT SUM(COST) FROM PRODUCT_MAST WHERE QTY>3 GROUP BY COMPANY;

Output:

```
Com1    150

Com2    170
```

**SUM( ) with HAVING**

SELECT COMPANY, SUM(COST) FROM PRODUCT_MAST GROUP BY COMPANY HAVING SUM(COST)>=170;

Output:

```
Com1    335

Com3    170
```

**3. <u>AVG function</u>**

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG( )

or

AVG( [ALL|DISTINCT] expression )

Example:

SELECT AVG(COST) FROM PRODUCT_MAST;

Output:

67.00

4. **MAX Function**

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX( )

or

MAX( [ALL|DISTINCT] expression )

Example:

SELECT MAX(RATE) FROM PRODUCT_MAST;

30

5. **<u>MIN Function</u>**

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

MIN( )

or

MIN( [ALL|DISTINCT] expression )

Example:

```
SELECT MIN(RATE)  FROM PRODUCT_MAST;
```

Output:

```
10
```

## 10. SQL NULL

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax

The basic syntax of **NULL** while creating a table.

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.

A field with a NULL value is the one that has been left blank during the record creation.

Example

The NULL value can cause problems when selecting data. However, because when comparing an unknown value to any other value, the result is always unknown and not included in the results. You must use the **IS NULL** or **IS NOT NULL** operators to check for a NULL value.

Consider the following CUSTOMERS table having the records as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | RAMESH   |  32 | AGRA      |  2000.00 |
|  2 | DINESH   |  25 | Delhi     |  1500.00 |
|  3 | CHARLES  |  23 | CALCUTTTA |  2000.00 |
|  4 | DINESH   |  25 | Mumbai    |  6500.00 |
|  5 | GANESH   |  27 | Bhopal    |  8500.00 |
|  6 | KAMAL    |  22 | CHENNAI   |          |
|  7 | VIMAL    |  24 | BANGLORE  |          |
+----+----------+-----+-----------+----------+
```

**IS NOT NULL:**

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
   FROM CUSTOMERS
   WHERE SALARY IS NOT NULL;
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | AGRA                  |  2000.00 |
|  2 | DINESH   |  25 | Delhi                  |  1500.00 |
|  3 | CHARLES  |  23 | CALCUTTA        |  2000.00 |
|  4 | DINESH   |  25 | Mumbai                |  6500.00 |
|  5 | GANESH   |  27 | Bhopal                |  8500.00 |
+----+----------+-----+-----------+----------+
```

**IS NULL:**

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
  FROM CUSTOMERS
  WHERE SALARY IS NULL;
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME      | AGE  | ADDRESS   | SALARY |
+----+----------+-----+-----------+----------+
|  6 | KAMAL     |  22    | CHENNAI    |             |
|  7 | MAHESH   |  24    | BANGLORE |             |
+----+----------+-----+-----------+----------+
```

## 11.SQL Sub Query

A Sub query is a query within another SQL query and embedded within the WHERE clause.

**Important Rule:**

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Sub queries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

### 1. Sub queries with the Select Statement

SQL subqueries are most frequently used with the Select statement.

**Syntax**

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
( SELECT column_name  from table_name WHERE ... );
```

**Example**

Consider the EMPLOYEE table have the following records:

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | John | 20 | US | 2000.00 |
| 2 | Stephan | 26 | Dubai | 1500.00 |
| 3 | David | 27 | Bangkok | 2000.00 |
| 4 | ARJUN | 29 | UK | 6500.00 |
| 5 | Kathrin | 34 | Bangalore | 8500.00 |
| 6 | Harry | 42 | China | 4500.00 |
| 7 | Jackson | 25 | Mizoram | 10000.00 |

The sub query with a SELECT statement will be:

```
SELECT * FROM EMPLOYEE
WHERE ID IN (SELECT ID  FROM EMPLOYEE
 WHERE SALARY > 4500);
```

Result:

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 4 | ARJUN | 29 | UK | 6500.00 |
| 5 | Kathrin | 34 | Bangalore | 8500.00 |

|  |  |  |  |  |
|---|---|---|---|---|
| 7 | Jackson | 25 | Mizoram | 10000.00 |

## 2. Subqueries with the INSERT Statement

- SQL sub query can also be used with the Insert statement. In the insert statement, data returned from the sub query is used to insert into another table.
- In the sub query, the selected data can be modified with any of the character, date functions.

**Syntax:**

```
INSERT INTO table_name (column1, column2, column3....)
SELECT * FROM table_name  WHERE VALUE OPERATOR
```

**Example**

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYEE_BKP
  SELECT * FROM EMPLOYEE
WHERE ID IN (SELECT ID   FROM EMPLOYEE);
```

## 3. Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

**Syntax**

```
UPDATE table  SET column_name = new_value  WHERE VALUE OPERATOR
(SELECT COLUMN_NAME
  FROM TABLE_NAME WHERE condition);
```

**Example**

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 HARIes in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
UPDATE EMPLOYEE  SET SALARY = SALARY * 0.25 WHERE AGE IN (SE
LECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | John | 20 | US | 2000.00 |
| 2 | Stephan | 26 | Dubai | 1500.00 |
| 3 | David | 27 | Bangkok | 2000.00 |

|  |  |  |  |  |
|---|---|---|---|---|
| 4 | Alina | 29 | UK | 1625.00 |
| 5 | Kathrin | 34 | Bangalore | 2125.00 |
| 6 | Harry | 42 | China | 1125.00 |
| 7 | Jackson | 25 | Mizoram | 10000.00 |

## 4. Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

**Syntax**

```
DELETE FROM TABLE_NAME WHERE VALUE OPERATOR  (SELECT CO
LUMN_NAME FROM TABLE_NAME
 WHERE condition);
```

**Example**

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

```
DELETE FROM EMPLOYEE  WHERE AGE IN (SELECT AGE FROM EMPLO
YEE_BKP WHERE AGE >= 29 );
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | John | 20 | US | 2000.00 |
| 2 | Stephan | 26 | Dubai | 1500.00 |
| 3 | David | 27 | Bangkok | 2000.00 |
| 7 | Jackson | 25 | Mizoram | 10000.00 |

## 12.SQL VIEWS:

- A view is nothing more than a SQL statement that is stored in the database with an associated name.
- A view is actually a composition of a table in the form of a predefined SQL query.
- A view can contain all rows of a table or select rows from a table.
- A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

***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];
```

Example:

Consider the CUSTOMERS table having the following records

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Arun | 25 | Delhi | 1500.00 |
| 3 | Balaji | 23 | Calcutta | 2000.00 |
| 4 | Charles | 25 | Mumbai | 6500.00 |
|  | Hari | 27 | Banglore | 8500.00 |

|  |  |  |  |  |
|---|---|---|---|---|
| 5 |  |  |  |  |
| 6 | Kamal | 22 | CHENNAI | 4500.00 |
| 7 | Mahesh | 24 | Andra | 10000.00 |

To create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

SQL > CREATE VIEW CUSTOMERS_VIEW

SELECT name, age FROM CUSTOMERS;

**OUTPUT** :

SQL > SELECT * FROM CUSTOMERS_VIEW;

| ID | NAME | AGE |
|---|---|---|
| 1 | Ramesh | 32 |
| 2 | Arun | 25 |
| 3 | Balaji | 23 |
| 4 | Charles | 25 |
| 5 | Hari | 27 |
|  | Kamal | 22 |

|  |  |  |
|---|---|---|
| 6 |  |  |
| 7 | Mahesh | 24 |

**Updating a View:**

Example: QUERY to update the age of Ramesh:

SQL > UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name='Ramesh';

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |
| 2 | Arun | 25 | Delhi | 1500.00 |
| 3 | Balaji | 23 | Calcutta | 2000.00 |
| 4 | Charles | 25 | Mumbai | 6500.00 |
| 5 | Hari | 27 | Banglore | 8500.00 |
| 6 | Kamal | 22 | CHENNAI | 4500.00 |
| 7 | Mahesh | 24 | Andra | 10000.00 |

## Inserting Rows into a View:

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we cannot insert rows in CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

## Deleting Rows into a View:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands  apply to the DELETE command.

Following is an example to delete a record having AGE= 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW    WHERE age = 22;
```

This would  delete a row from the base table CUSTOMERS and same would reflect in the view itself.

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |
| 2 | Arun | 25 | Delhi | 1500.00 |
| 3 | Balaji | 23 | Calcutta | 2000.00 |
| 4 | Charles | 25 | Mumbai | 6500.00 |
|  | Hari | 27 | Banglore | 8500.00 |

|  |  |  |  |  |
|---|---|---|---|---|
| 5 |  |  |  |  |
| 7 | Mahesh | 24 | Andra | 10000.00 |

**Dropping Views:**

To drop CUSTOMERS_VIEW from CUSTOMERS table:

DROP VIEW CUSTOMERS_VIEW

## 13.Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example:

**EMPLOYEE**

| EMP_CODE | EMP_NAME |
|---|---|
| 101 | Stephan |
| 102 | Jack |
| 103 | Harry |

**SALARY**

| EMP_CODE | SALARY |
|---|---|
| 101 | 50000 |
| 102 | 30000 |
| 103 | 25000 |

Operation: (EMPLOYEE ⋈ SALARY)

| EMP_CODE | EMP_NAME | SALARY |
|---|---|---|
| 101 | Stephan | 50000 |
| 102 | Jack | 30000 |
| 103 | Harry | 25000 |

**Types of Join operations**:

1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by ⋈.

**Example:** Let's use the above EMPLOYEE table and SALARY table:

**Input:** EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

**Output:**

| EMP_NAME | SALARY |
|---|---|
| Stephan | 50000 |
| Jack | 30000 |
| Harry | 25000 |

## 2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

**Example:**

**EMPLOYEE**

| EMP_NAME | STREET | CITY |
|---|---|---|
| Ram | Civil line | Mumbai |
| Shyam | Park street | Kolkata |
| Ravi | M.G. Street | Delhi |
| Hari | Nehru nagar | Hyderabad |

**FACT_WORKERS**

| EMP_NAME | BRANCH | SALARY |
|---|---|---|
| Ram | Infosys | 10000 |
| Shyam | Wipro | 20000 |
| Kuber | HCL | 30000 |
| Hari | TCS | 50000 |

**Input:** (EMPLOYEE $\bowtie$ FACT_WORKERS)

**Output:**

| EMP_NAME | STREET | CITY | BRANCH | SALARY |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru nagar | Hyderabad | TCS | 50000 |

An outer join is basically of three types:

a. Left outer join

b. Right outer join

c. Full outer join

## a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by $\bowtie$ .

**Example:** EMPLOYEE

| EMP_NAME | STREET | CITY |
|---|---|---|
| Ram | Civil line | Mumbai |
| Shyam | Park street | Kolkata |
| Ravi | M.G. Street | Delhi |
| Hari | Nehru nagar | Hyderabad |

FACT_WORKERS

| EMP_NAME | BRANCH | SALARY |
|---|---|---|
| Ram | Infosys | 10000 |
| Shyam | Wipro | 20000 |
| Kuber | HCL | 30000 |
| Hari | TCS | 50000 |

**Input:**

EMPLOYEE ⋈ FACT_WORKERS

| EMP_NAME | STREET | CITY | BRANCH | SALARY |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |

### b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by .

**Example:**

**Input:** EMPLOYEE ⋈ FACT_WORKERS

**Output:**

| EMP_NAME | BRANCH | SALARY | STREET | CITY |
|---|---|---|---|---|
| Ram | Infosys | 10000 | Civil line | Mumbai |
| Shyam | Wipro | 20000 | Park street | Kolkata |
| Hari | TCS | 50000 | Nehru street | Hyderabad |
| Kuber | HCL | 30000 | NULL | NULL |

c. Full outer join:

Full outer join is like a left or right join except that it contains all rows from both tables.

In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.

It is denoted by .

**Example:**

**Input:** EMPLOYEE $\bowtie$ FACT_WORKERS

**Output:**

| EMP_NAME | STREET | CITY | BRANCH | SALARY |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |
| Kuber | NULL | NULL | HCL | 30000 |

## 3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

**Example:**

**CUSTOMER RELATION**

| CLASS_ID | NAME |
|---|---|
| 1 | John |
| 2 | Harry |
| 3 | Jackson |

**PRODUCT**

| PRODUCT_ID | CITY |
|---|---|
| 1 | Delhi |
| 2 | Mumbai |
| 3 | Noida |

**Input:** CUSTOMER ⋈ PRODUCT

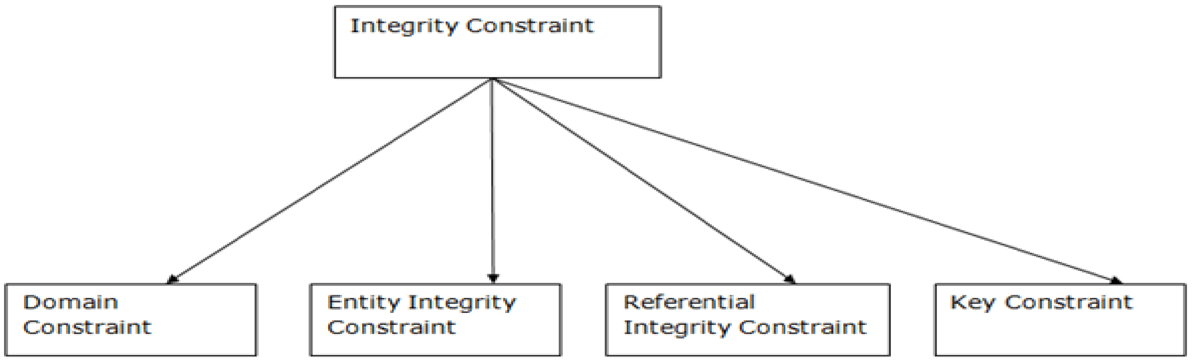
**Output:**

| CLASS_ID | NAME | PRODUCT_ID | CITY |
|---|---|---|---|
| 1 | John | 1 | Delhi |
| 2 | Harry | 2 | Mumbai |
| 3 | Harry | 3 | Noida |

## **14.Integrity Constraints**

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

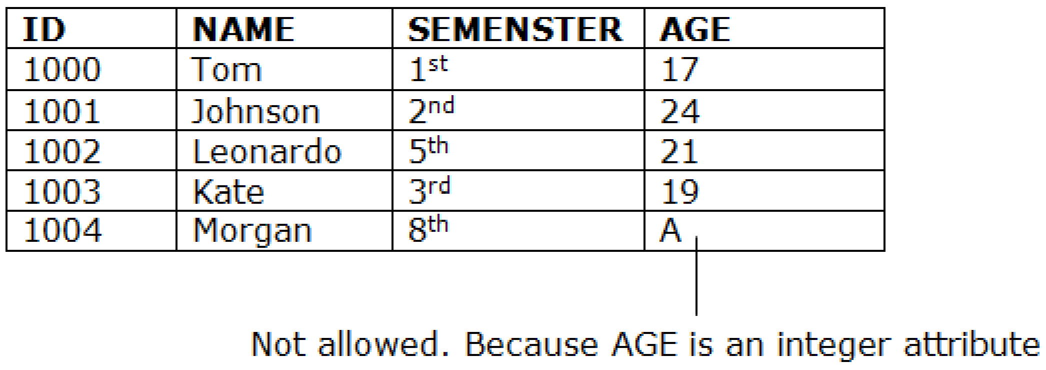
### Types of Integrity Constraint



### 1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, date, currency, etc. The value of the attribute must be available in the corresponding domain.

**Example:**



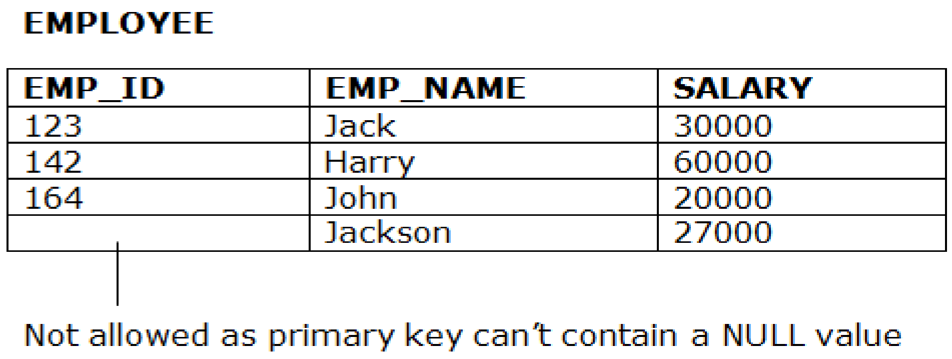
| ID | NAME | SEMENSTER | AGE |
|---|---|---|---|
| 1000 | Tom | 1st | 17 |
| 1001 | Johnson | 2nd | 24 |
| 1002 | Leonardo | 5th | 21 |
| 1003 | Kate | 3rd | 19 |
| 1004 | Morgan | 8th | A |

Not allowed. Because AGE is an integer attribute

## 2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

**Example:**



EMPLOYEE

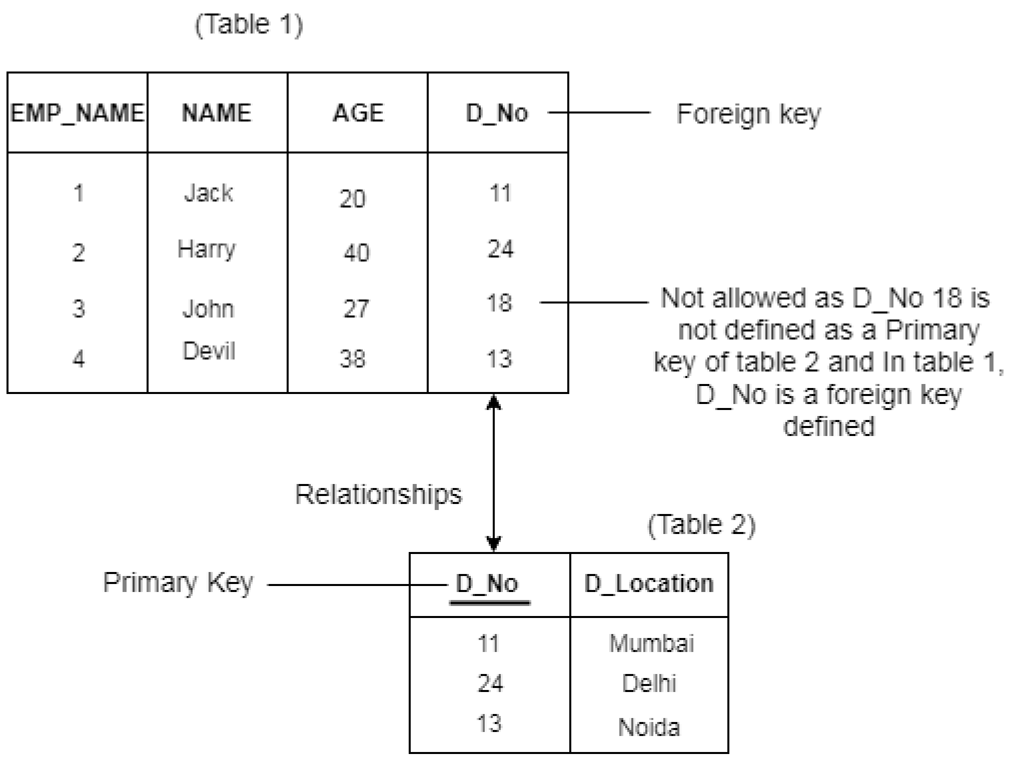
| EMP_ID | EMP_NAME | SALARY |
|---|---|---|
| 123 | Jack | 30000 |
| 142 | Harry | 60000 |
| 164 | John | 20000 |
|  | Jackson | 27000 |

Not allowed as primary key can't contain a NULL value

## 3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

**Example:**


(Table 1)
Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
(Table 2)
Primary Key

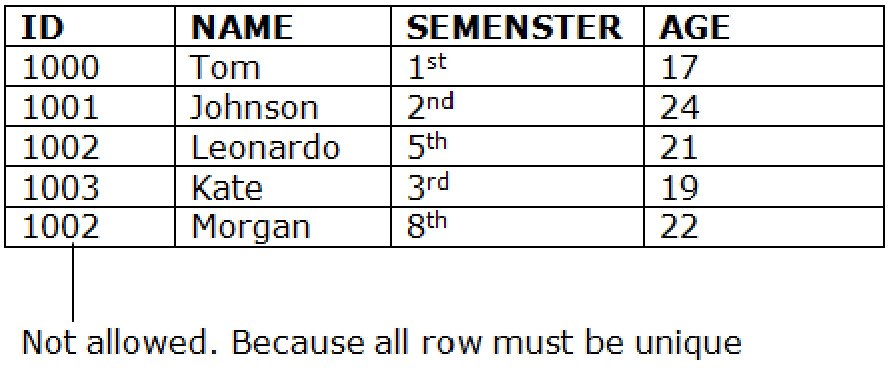
| EMP_NAME | NAME | AGE | D_No |
|---|---|---|---|
| 1 | Jack | 20 | 11 |
| 2 | Harry | 40 | 24 |
| 3 | John | 27 | 18 |
| 4 | Devil | 38 | 13 |

| D_No | D_Location |
|---|---|
| 11 | Mumbai |
| 24 | Delhi |
| 13 | Noida |

## 4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

**Example:**



| ID | NAME | SEMENSTER | AGE |
|---|---|---|---|
| 1000 | Tom | 1st | 17 |
| 1001 | Johnson | 2nd | 24 |
| 1002 | Leonardo | 5th | 21 |
| 1003 | Kate | 3rd | 19 |
| 1002 | Morgan | 8th | 22 |

Not allowed. Because all row must be unique

# UNIT -4

# RELATIONAL DATA BASE DESIGN

## 1.What is Database Design?

- **Database Design** is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems.
- Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space.
- The database designer decides how the data elements are relates with each others and what data must be stored.

The main objectives of database designing are to produce logical and physical designs models of the proposed database system.

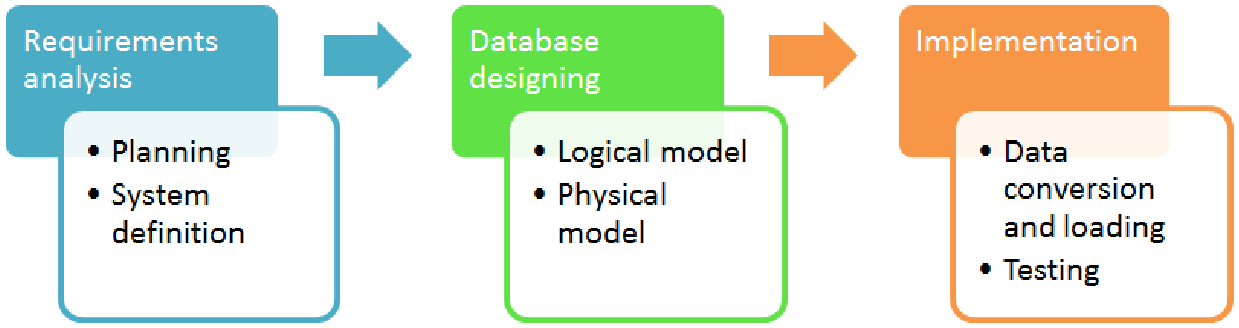
The logical model concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.

The physical data design model involves translating the logical design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

## Importance of design:

- It helps to meet the requirements of the users
- Using database design the system have high performance.
- Database designing is crucial to high performance database system.

**<u>Database development life cycle</u>**



**Requirements analysis**

- **Planning** :

  This stages concerns with planning of entire Database Development Life Cycle. It takes into consideration the Information Systems strategy of the organization.

- **System definition** :

  This stage defines the scope and boundaries of the proposed database system.

**Database designing**

- **Logical model** :

  This stage is concerned with developing a database model based on requirements. The entire design is on paper without any physical implementations or specific DBMS considerations.

- **Physical model** :

  This stage implements the logical model of the database taking into account the DBMS and physical implementation factors.

## Implementation

- **Data conversion and loading** :

  This stage is concerned with importing and converting data from the old system into the new database.

- **Testing** :

  This stage is concerned with the identification of errors in the newly implemented system .It checks the database against requirement specifications.

## 2. **RELATIONAL DATA BASE DESIGN**

Relational database design (RDD) models information and data into a set of tables with rows and columns. Each row of a relation/table represents a record, and each column represents an attribute of data.

The Structured Query Language (SQL) is used to manipulate relational databases. The design of a relational database is composed of four stages, where the data are modeled into a set of related tables. The stages are:

- Define relations/attributes
- Define primary keys
- Define relationships
- Normalization

### ***Relational Database Design (RDD)***

- ➢ Relational databases differ from other databases in their approach to organizing data and performing transactions.
- ➢ In an RDD, the data are organized into tables and all types of data access are carried out via controlled transactions.
- ➢ Relational database design satisfies the ACID (atomicity, consistency, integrity and durability) properties required from a database design.
- ➢ Relational database design mandates the use of a database server in applications for dealing with data management problems.

The four stages of an RDD are as follows:

- Relations and attributes:

  The various tables and attributes related to each table are identified. The tables represent entities, and the attributes represent the properties of the respective entities.

- Primary keys:

  The attribute or set of attributes that help in uniquely identifying a record is identified and assigned as the primary key.

- Relationships:

  The relationships between the various tables are established with the help of foreign keys. Foreign keys are attributes occurring in a table that are primary keys of another table. The types of relationships that can exist between the relations (tables) are:

  - One to one
  - One to many
  - Many to many

Relationships can be understand easily by using, an entity-relationship diagram, which can be used to describe the entities, their attributes and the relationship in a diagrammatic way.

## 3. Entity-Relationship Diagrams (or) E-R Diagram:

Entity relationship diagram displays the relationships of entity set stored in a database. ER diagrams help you to explain the logical structure of databases.

An ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

- ER model allows you to draw Database Design.
- It is an easy to use graphical tool for modeling data.
- Widely used in Database Design.
- It is a GUI representation of the logical structure of a Database.
- It helps you to identify the entities which exist in a system and the relationships between those entities.

### Applications of E-R Diagram:

The main reasons for using the ER Diagram

- Helps you to define terms related to entity relationship modeling.
- Provide a preview of how all your tables should connect, what fields are going to be on each table.
- Helps to describe entities, attributes, relationships.
- ER diagrams are translatable into relational tables which allow you to build databases quickly.
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications.
- The database designer gains a better understanding of the information to be contained in the database with the help of ER diagram.
- ERD is allowed you to communicate with the logical structure of the database to users

## Components of Entity Relationship Model

ER Model is used to model the logical view of the system from data perspective which consists of these components:

- ➢ Entity
- ➢ Entity Type,
- ➢ Entity Set

**ENTITY:**

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

**Examples of entities:**

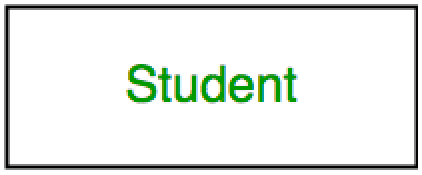
- **Person:** Employee, Student, Patient

**Entity set:**

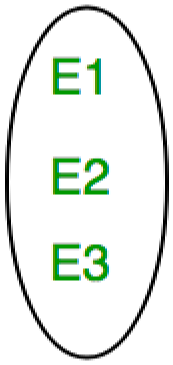
An entity set is a group of similar kind of entities. It may contain entities with attribute sharing similar values. Entities are represented by their properties, which also called attributes. All attributes have their separate values.

For example,

A student entity may have a name, age, class, as attributes.



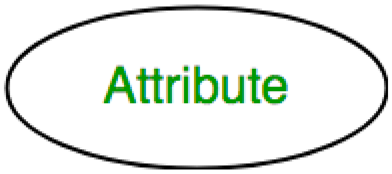
Entity Type



Entity Set

**<u>Attribute(s):</u>**

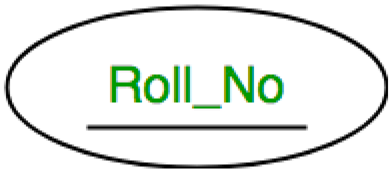
Attributes are the properties which define the entity type. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.



1.***Key Attribute*** :

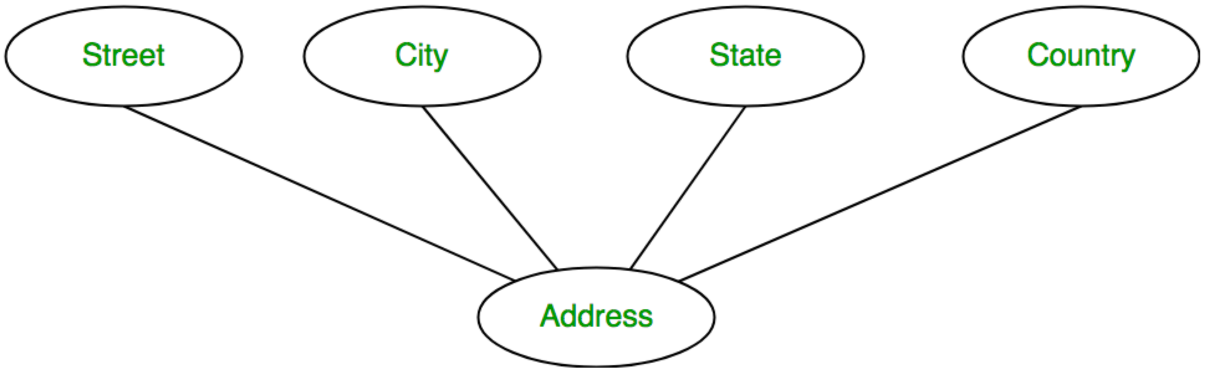
The attribute which uniquely identifies each entity in the entity set is called key attribute.

For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.



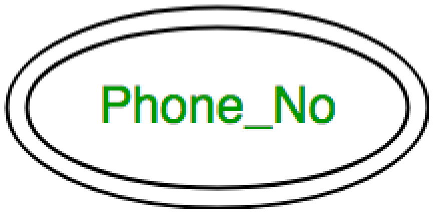
2.***Composite Attribute*** :

An attribute composed of many other attribute is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.



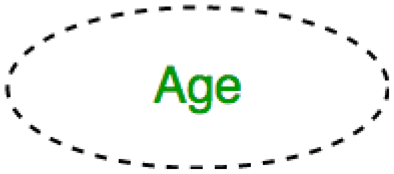
3.***Multivalued Attribute*** :

An attribute consisting more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

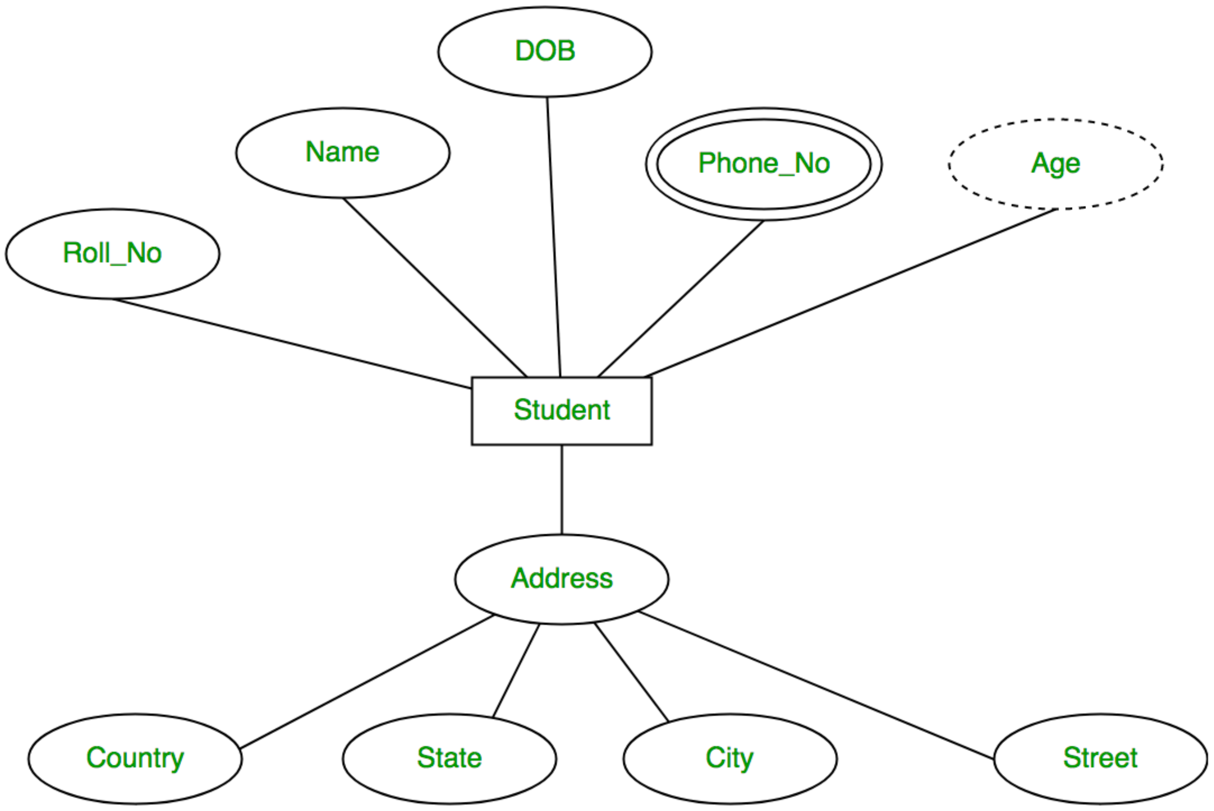


4.***Derived Attribute*** :

An attribute which can be derived from other attributes of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.



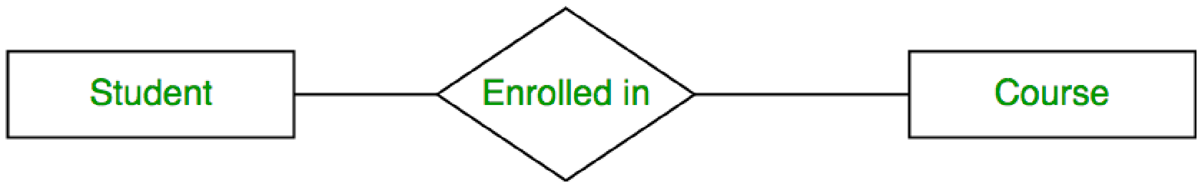
The complete entity type Student with its attributes can be represented as:



**Relationship Type** :

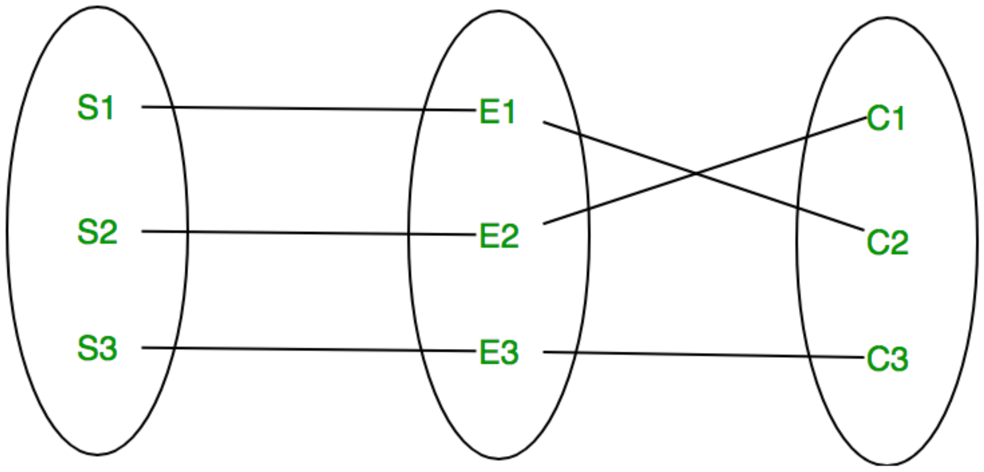
A relationship type represents the association between entity types.

For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.



**Relationship Set**:

A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.

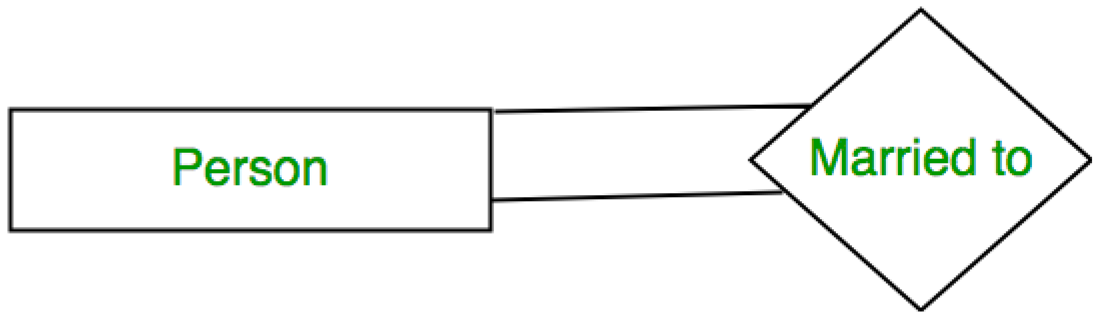


***<u>Degree of a relationship set</u>*:**

The number of different entity sets participating in a relationship set is called as degree of a relationship set.

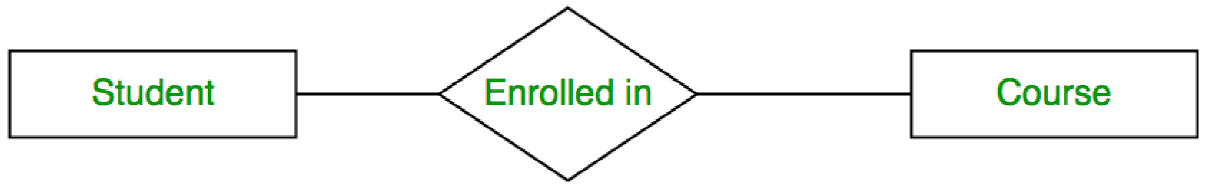
1.***Unary Relationship*** :

When there is only ONE entity set participating in a relation, the relationship is called as unary relationship. For example, one person is married to only one person.



2.***Binary Relationship*** :

When there are TWO entities set participating in a relation, the relationship is called as binary relationship.For example, Student is enrolled in Course.



3.***n-ary Relationship*** :

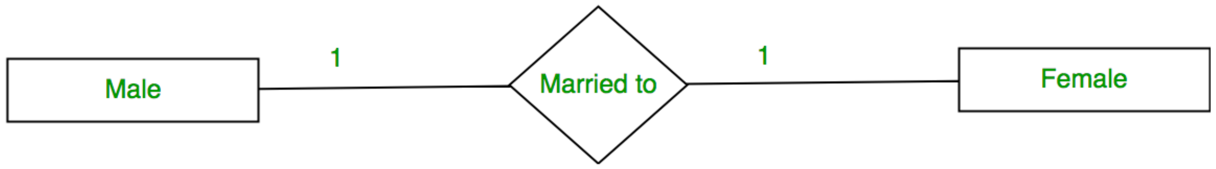
When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

## Cardinality:

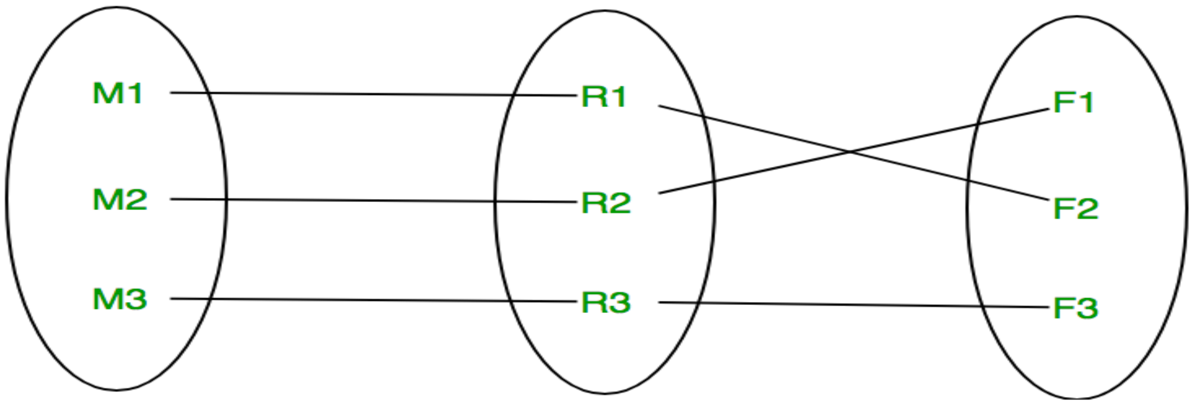
The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

1.***One to one*** :

When each entity in each entity set can take part only once in the relationship, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.



Using Sets, it can be represented as:

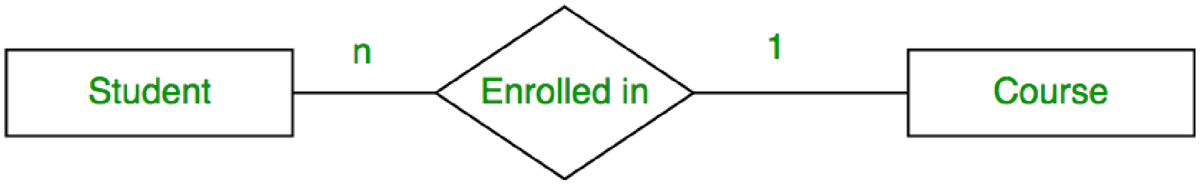


2.***Many to one :***

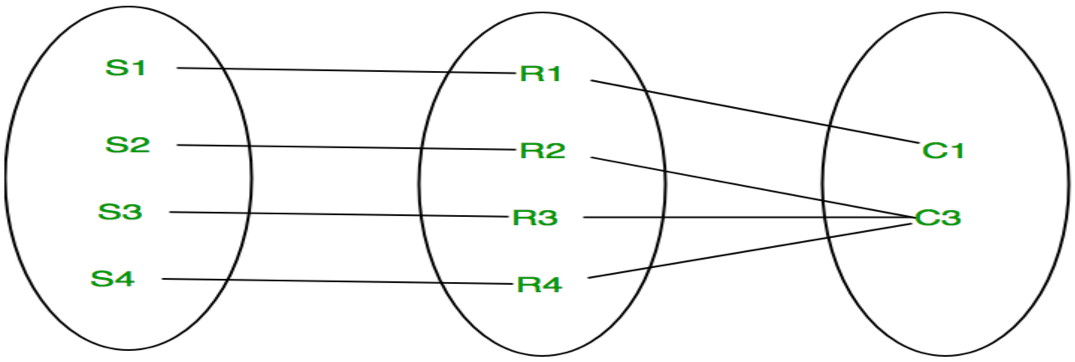
When entities in one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one.

Example:

A student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.



Using Sets, it can be represented as:



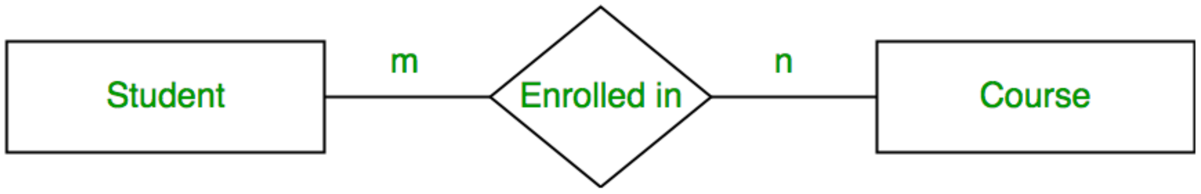
In this case, each student is taking only 1 course but 1 course has been taken by many students.

3.***Many to many***:

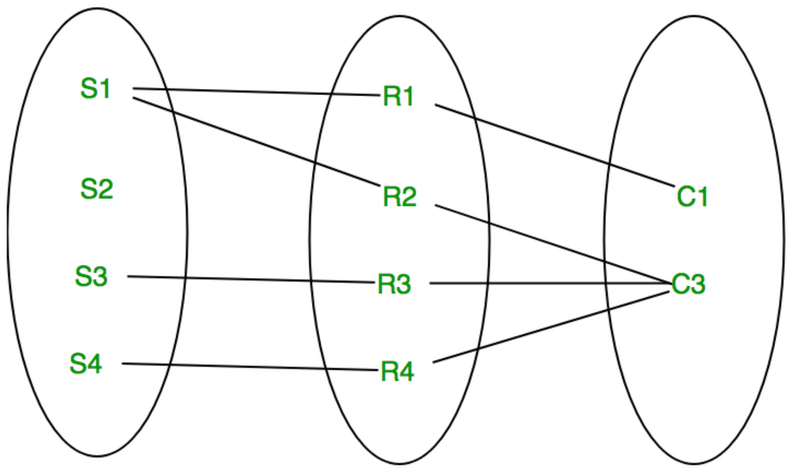
When entities in all entity sets can take part more than once in the relationship cardinality is many to many.

Example:

A student can take more than one course and one course can be taken by many students. So the relationship will be many to many.



Using sets, it can be represented as:



**<u>Participation Constraint:</u>**

Participation Constraint is applied on the entity participating in the relationship set.

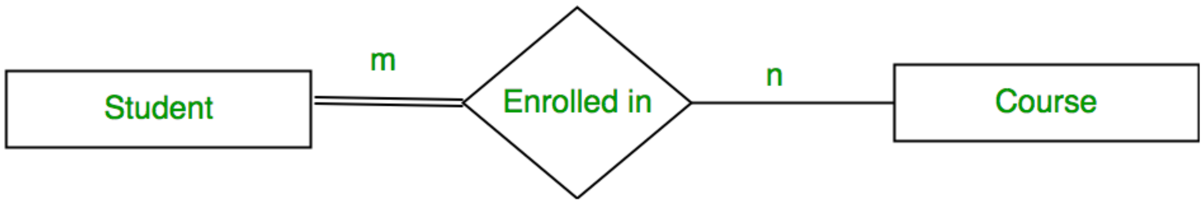
1.***Total Participation***:

Each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

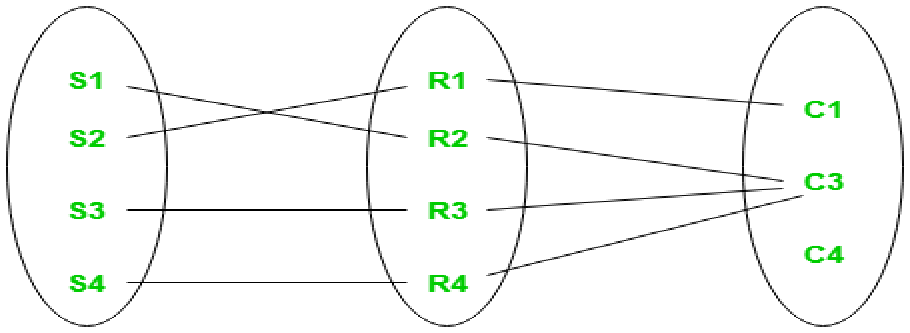
2.***Partial Participation*** :

The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Using set, it can be represented as,



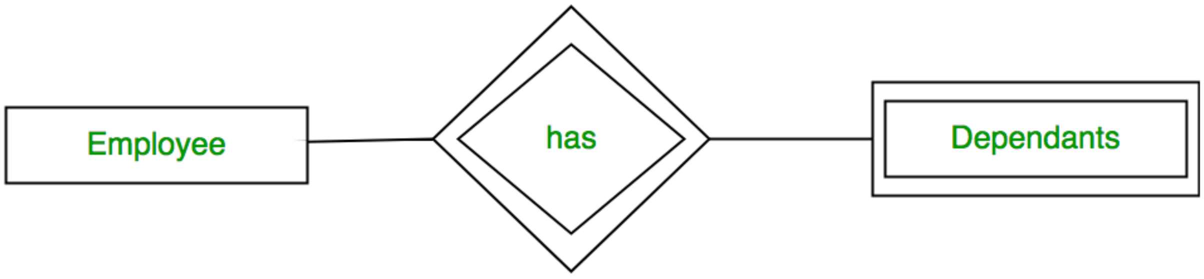
**<u>Weak Entity Type and Identifying Relationship</u>**:

An entity type has a key attribute which uniquely identifies each entity in the entity set. Some entity type for which key attribute can't be defined. These are called Weak Entity type.

A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond.

For example,

A company may store the information of dependants (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependant.



## **4.DATA ABSTRACTION:**

Generalization, Specialization and Aggregation in ER model are used for data abstraction in which abstraction mechanism is used to hide details of a set of objects.

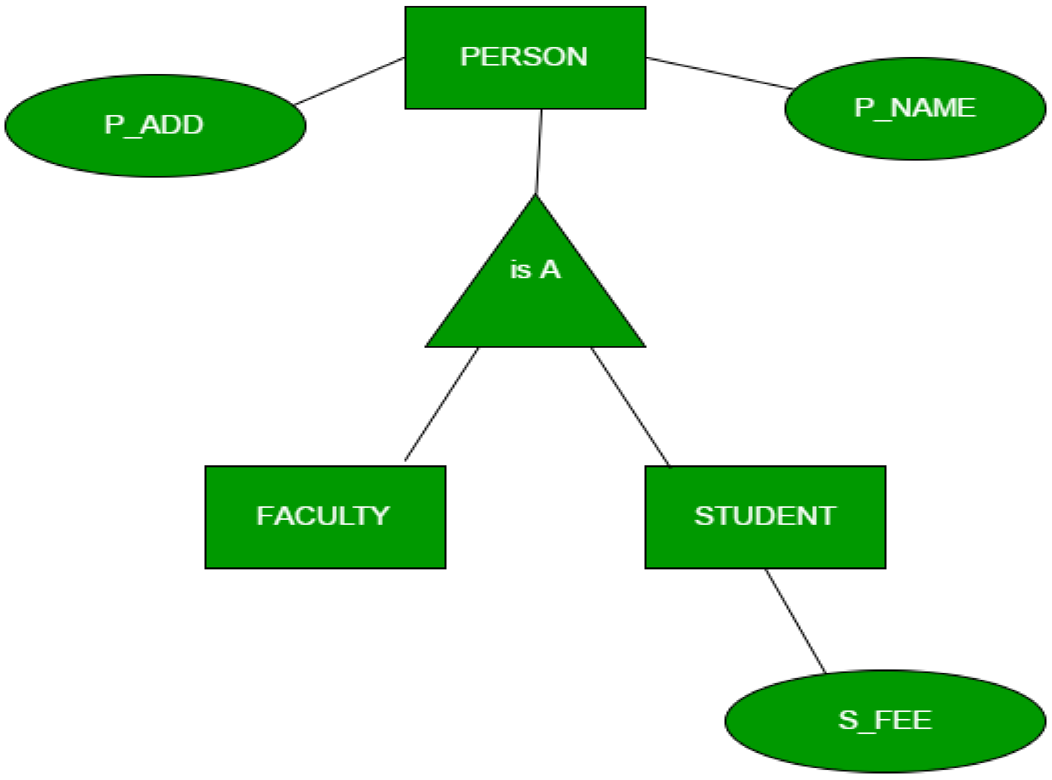
***Generalization :***

Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it.

It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common.

For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure.

In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity(STUDENT).

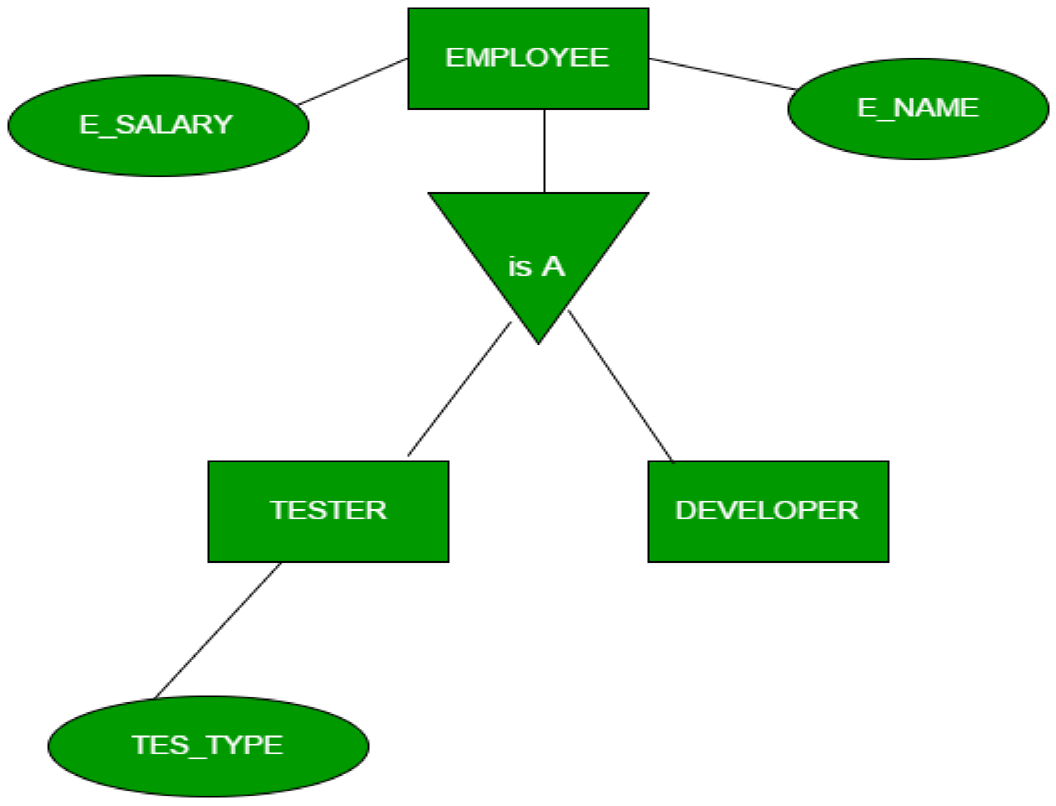


***Specialization* :**

In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities.

For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure.

In this case, common attributes like E_NAME, E_SAL etc. become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).

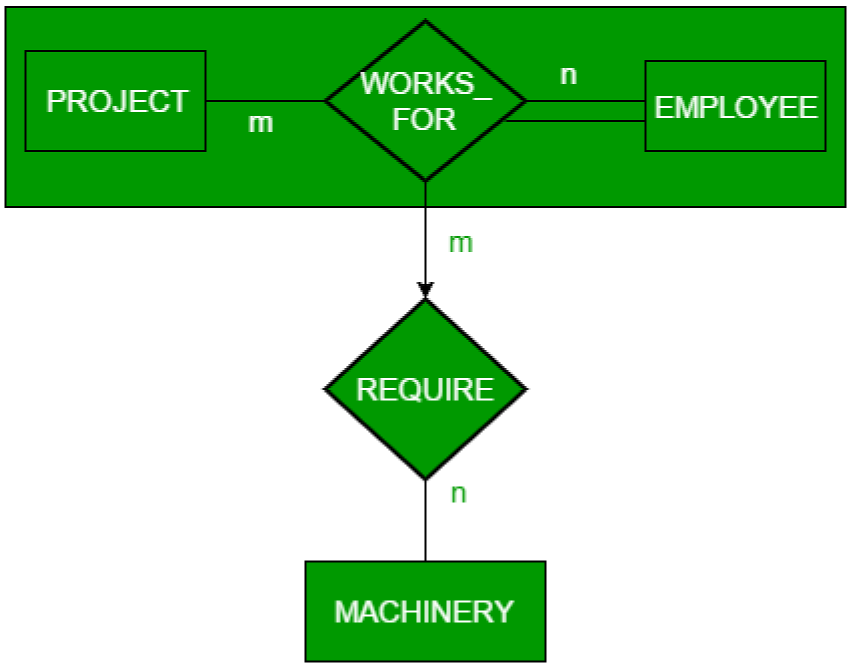


Specialization

***Aggregation :***

An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity.

For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY



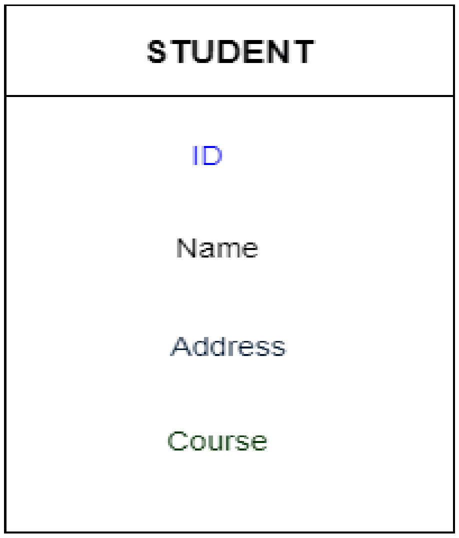
Aggregation

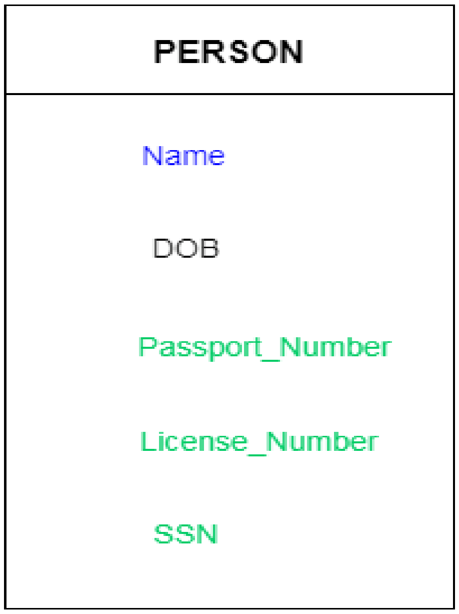
## 5.DATABASE KEYS

### Keys

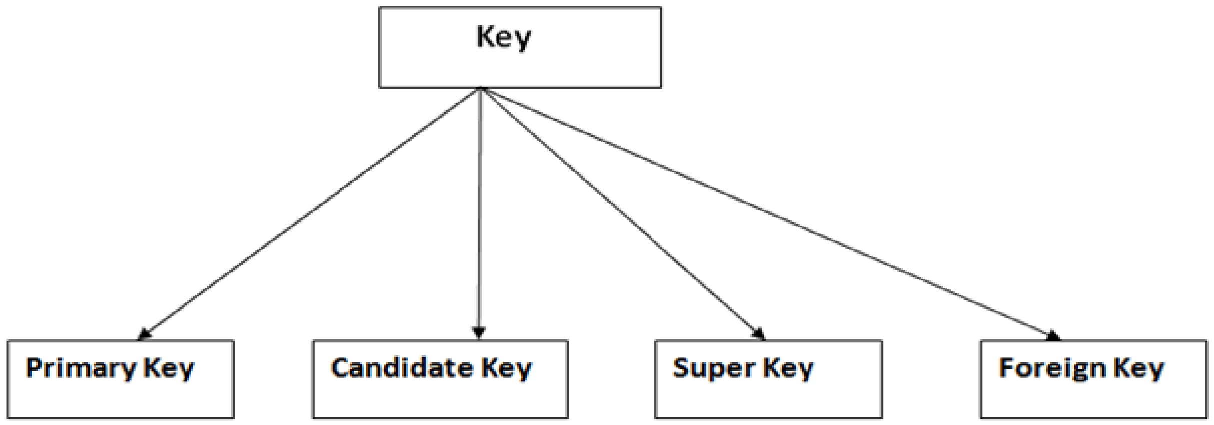
- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

**For example:** In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.





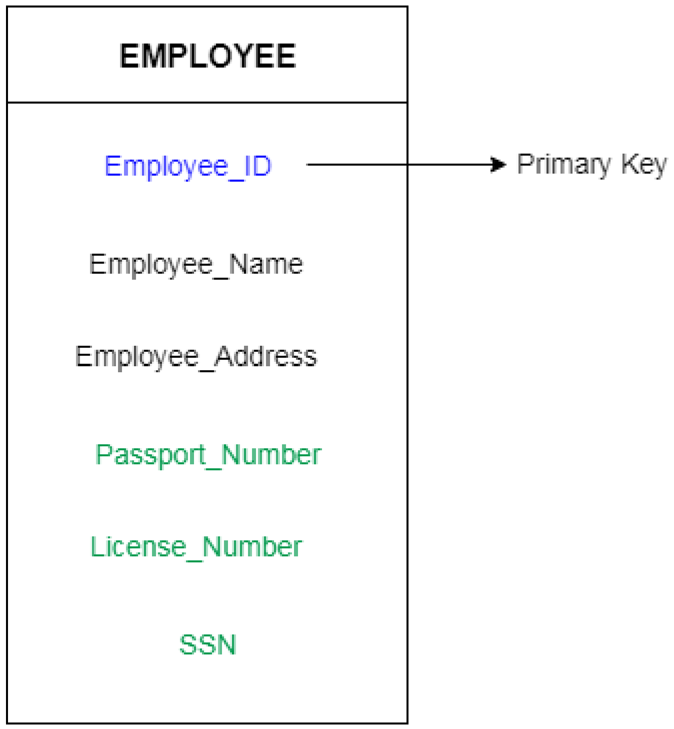
***Types of keys***:



## 1. *Primary key*

Primary key  is the first key which is used to identify one and only one instance of an entity uniquely.

- An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.



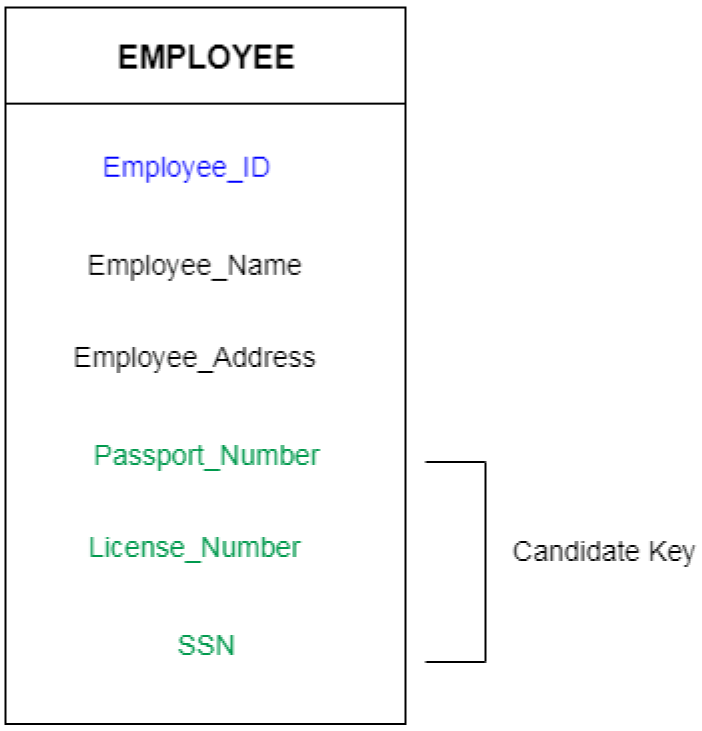
## 2. *Candidate key*

A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.

- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

**For example:**

In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.



## 3. Super Key

Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

**For example:**

In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

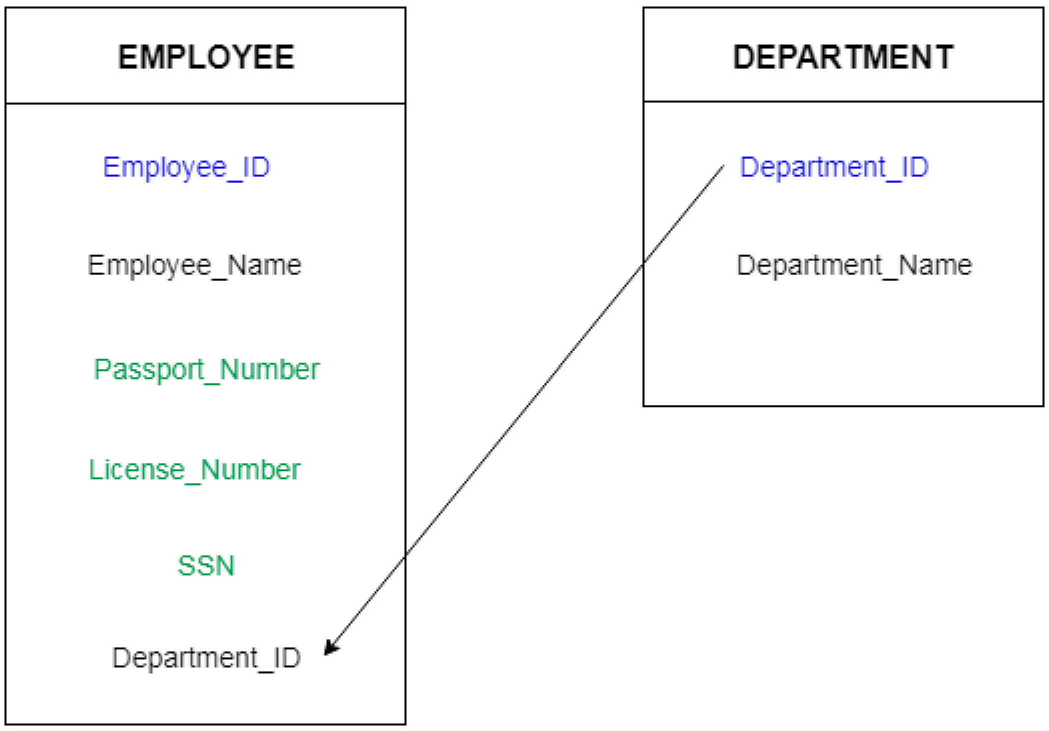
## 4. Foreign key

Foreign keys are the column of the table which is used to point to the primary key of another table.

Example:

In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.

- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



# Unit-5

## NORMALIZATION

### Normalization in DBMS:

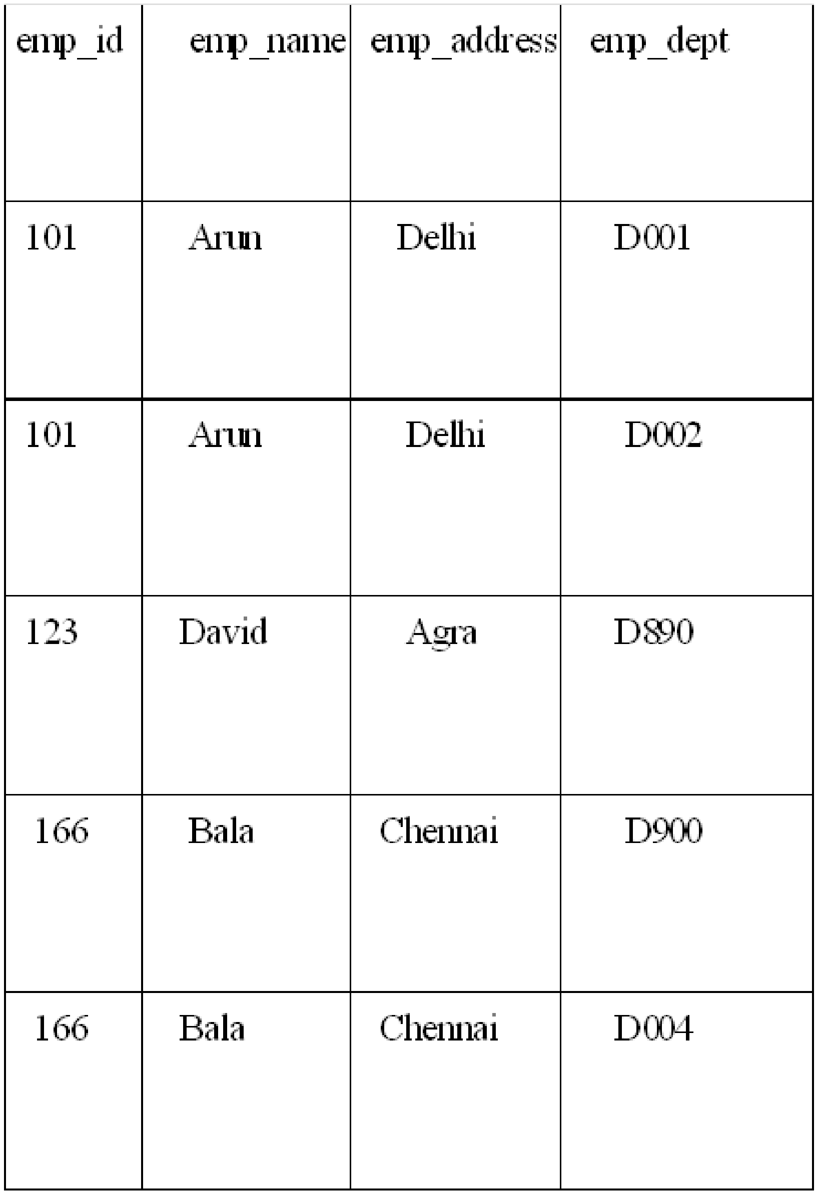
**Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

### Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

**Example**:

a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee’s id, emp_name for storing employee’s name, emp_address for storing employee’s address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:



| emp_id | emp_name | emp_address | emp_dept |
|---|---|---|---|
| 101 | Arun | Delhi | D001 |
| 101 | Arun | Delhi | D002 |
| 123 | David | Agra | D890 |
| 166 | Bala | Chennai | D900 |
| 166 | Bala | Chennai | D004 |

The above table is not normalized. We will see the problems that we face when a table is not normalized.

**Update anomaly**:

In the above table we have two rows for employee Arun as he belongs to two departments of the company. If we want to update the address of Arun then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Arun would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**:

Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn’t allow nulls.

**Delete anomaly**:

Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee David since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

## Normalization

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

## First normal form (1NF)

An attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

**Example**: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

| emp_id | emp_name | emp_address | emp_mobile |
|---|---|---|---|
| 101 | Hari | New Delhi | 8912312390 |
| 102 | John | Kanpur | 8812121212<br><br>9900012222 |
| 103 | Ravi | Chennai | 7778881212 |
| 104 | Ram | Bangalore | 9990000123<br>8123450987 |

Two employees (John & Ram) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says "each attribute of a table must have atomic (single) values", the emp_mobile values for employees John & Ram violates that rule.

To make the table complies with 1NF we should have the data like this:

| emp_id | emp_name | emp_address | emp_mobile |
|---|---|---|---|
| 101 | Hari | New Delhi | 8912312390 |
| 102 | John | Kanpur | 8812121212 |
| 102 | John | Kanpur | 9900012222 |
| 103 | Ravi | Chennai | 7778881212 |
| 104 | Ram | Bangalore | 9990000123 |
| 104 | Ram | Bangalore | 8123450987 |

## 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**:

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

To make the table complies with 2NF we can break it in two tables like this:
**teacher_details table:**

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

**teacher_subject table:**

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

Now the tables comply with Second normal form (2NF).

## Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

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

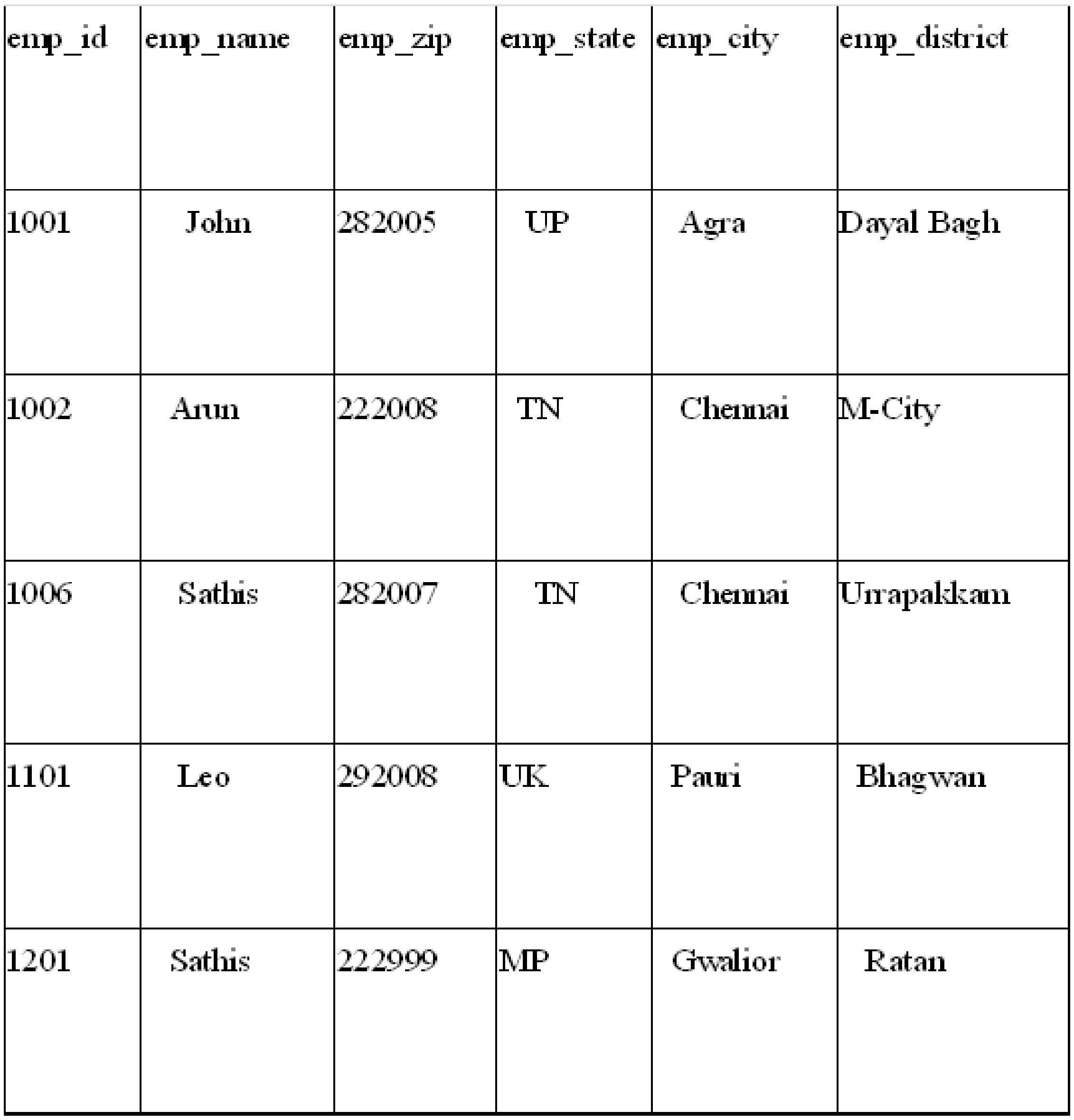
In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

**Example**:

Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:



| emp_id | emp_name | emp_zip | emp_state | emp_city | emp_district |
|---|---|---|---|---|---|
| 1001 | John | 282005 | UP | Agra | Dayal Bagh |
| 1002 | Arun | 222008 | TN | Chennai | M-City |
| 1006 | Sathis | 282007 | TN | Chennai | Urrapakkam |
| 1101 | Leo | 292008 | UK | Pauri | Bhagwan |
| 1201 | Sathis | 222999 | MP | Gwalior | Ratan |

**Super keys**: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
**Candidate Keys**: {emp_id}
**Non-prime attributes**: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

**employee table:**

| emp_id | emp_name | emp_zip |
|---|---|---|
| 1001 | John | 282005 |
| 1002 | Arun | 222008 |
| 1006 | Sathis | 282007 |
| 1101 | Leo | 292008 |
| 1201 | Sathis | 222999 |

**employee_zip table:**

| emp_zip | emp_state | emp_city | emp_district |
|---|---|---|---|
| 282005 | UP | Agra | Dayal Bagh |
| 222008 | TN | Chennai | M-City |
| 282007 | TN | Chennai | Urrapakkam |
| 292008 | UK | Pauri | Bhagwan |
| 222999 | MP | Gwalior | Ratan |

## Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

**Example**: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

| emp_id | emp_nationality | emp_dept | dept_type | dept_no_of_emp |
|---|---|---|---|---|
| 1001 | Austrian | Production and planning | D001 | 200 |
| 1001 | Austrian | stores | D001 | 250 |
| 1002 | American | design and technical support | D134 | 100 |
| 1002 | American | Purchasing department | D134 | 600 |

**Functional dependencies in the table above**:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

**Candidate key**: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

**emp_nationality table:**

| emp_id | emp_nationality |
|---|---|
| 1001 | Austrian |
| 1002 | American |

**emp_dept table:**

| emp_dept | dept_type | dept_no_of_emp |
|---|---|---|
| Production and planning | D001 | 200 |
| stores | D001 | 250 |
| design and technical support | D134 | 100 |
| Purchasing department | D134 | 600 |

**emp_dept_mapping table:**

| emp_id | emp_dept |
|---|---|
| 1001 | Production and planning |
| 1001 | stores |
| 1002 | design and technical support |
| 1002 | Purchasing department |

**Functional dependencies**:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

**Candidate keys**:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

**Functional Dependency:**

**What is a Functional Dependency?**

**Functional Dependency (FD)** determines the relation of one attribute to another attribute in a database management system (DBMS) system. Functional dependency helps you to maintain the quality of data in the database.

A functional dependency is denoted by an arrow →. The functional dependency of X on Y is represented by X → Y. Functional Dependency plays a vital role to find the difference between good and bad database design.

**Example:**

| Employee number | Employee Name | Salary | City |
|---|---|---|---|
| 1 | Dana | 50000 | San Francisco |
| 2 | Francis | 38000 | London |
| 3 | Andrew | 25000 | Tokyo |

, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended .In this example, if we know the value of Employee number, we can obtain Employee Name on Employee number.

**Rules of Functional Dependencies**

Below given are the Three most important rules for Functional Dependency:

- Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y.
- Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

**Types of Functional Dependencies**

- Multivalued dependency:
- Trivial functional dependency:
- Non-trivial functional dependency:
- Transitive dependency:

**Multivalued dependency in DBMS**

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.

**Example:**

| **Car_model** | **Maf_year** | **Color** |
|---|---|---|
| H001 | 2017 | Metallic |
| H001 | 2017 | Green |
| H005 | 2018 | Metallic |
| H005 | 2018 | Blue |
| H010 | 2015 | Metallic |
| H033 | 2012 | Gray |

In this example, maf_year and color are independent of each other but dependent on car_model. In this example, these two columns are said to be multivalue dependent on car_model.

This dependence can be represented like this:

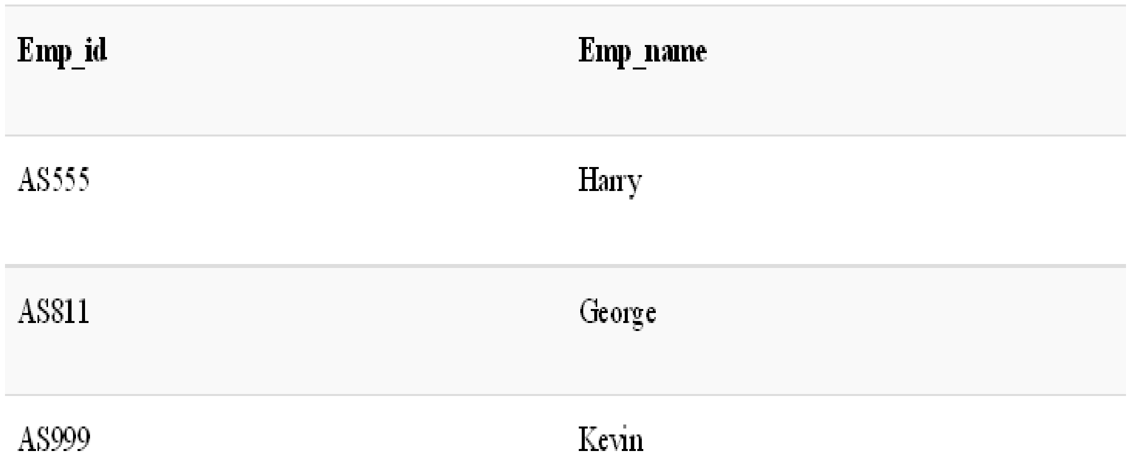
car_model -> maf_year and car_model-> colour.

**Trivial Functional dependency:**

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.

So, X -> Y is a trivial functional dependency if Y is a subset of X.

For example:



| Emp_id | Emp_name |
|---|---|
| AS555 | Harry |
| AS811 | George |
| AS999 | Kevin |

Consider this table with two columns Emp_id and Emp_name.

{Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

**Non trivial functional dependency :**

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

| **Company** | **CEO** | **Age** |
|---|---|---|
| Microsoft | Satya Nadella | 51 |
| Google | Sundar Pichai | 46 |
| Apple | Tim Cook | 57 |

**Example:**

(Company} -> {CEO} (if we know the Company, we knows the CEO name)

But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

**Transitive dependency:**

A transitive is a type of functional dependency which happens when it is indirectly formed by two functional dependencies.

**Example:**

| Company | CEO | Age |
|---|---|---|
| Microsoft | Satya Nadella | 51 |
| Google | Sundar Pichai | 46 |
| Alibaba | Jack Ma | 54 |

{Company} -> {CEO} (if we know the compay, we know its CEO's name)

{CEO } -> {Age} If we know the CEO, we know the Age

According to the rule of rule of transitive dependency:

{ Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.

Hints:Transitive dependency can only occur in a relation of three or more attributes.

**Advantages of Functional Dependency**

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database edsign