SKDAV GOVT. POLYTECHNIC ROURKELA

DATABASE MANAGEMENT SYSTEM

COURSE CONTENTS

1.0 BASIC CONCPETS OF DBMS

- 1.1 Purpose of database Systems
- 1.2 Explain Data abstraction
- 1.3 Database users
- 1.4 Data definition language
- 1.5 Data Dictionary

2.0 DATA MODELS

- 2.1 Data independence
- 2.2 Entity relationship models
- 2.3 Entity sets and Relationship sets
- 2.4 Explain Attributes
- 2.5 Mapping constraints
- 2.6 E-R Diagram
- 2.7 Relational model
- 2.8 Hierarchical model
- 2.9 Network model

3.0 RELATIONAL DATABASE

- 3.1 Relational algebra
- 3.2 Different operators select, project, join , simple Examples

4.0 NORMALIZATION IN RELATIONAL SYSTEM

- 4.1 Functional Dependencies
- 4.2 Lossless join
- 4.3 Importance of normalization
- 4.4 Compare First second and third normal forms
- 4.5 Explain BCNF

5.0 STRUCTURED QUERY LANGUAGE

- 5.1 Elementary idea of Query language
- 5.2 Queries in SQL
- 5.3 Simple queries to create, update, insert in SQL

6.0 TRANSACTION PROCESSING CONCEPTS

- 6.1 Idea about transaction processing
- 6.2 Transaction & system concept
- 6.3 Desirable properties of transaction
- 6.4 Schedules and recoverability

7.0 CONCURRENCY CONTROL CONCEPTS

- 7.1 Basic concepts,
- 7.2 Locks, Live Lock, Dead Lock,
- 7.3 Serializability (only fundamentals)

8.0 SECURITY AND INTEGRITY

- 8.1 Authorization and views
- 8.2 Security constraints
- 8.3 Integrity Constraints
- 8.4 Discuss Encryption

UNIT-1 : BASIC CONCEPTS OF DBMS

Data

- **Data** is an individual unit that contains raw materials which do not carry any specific meaning. It can be used in a variety of forms like text, numbers, media, bytes, etc. it can be stored in pieces of paper or electronic memory, etc. **Data** doesn't depend on **information**.
- Word 'Data' is originated from the word 'datum' that means 'single piece of information.' It is plural of the word datum.
- In computing, Data is information that can be translated into a form for efficient movement and processing. Data is interchangeable.

Information

- **Information** is a group of **data** that collectively carries a logical meaning. **Information** depends on **data**.

Database

- The database is a collection of inter-related data stored in a standardized format, capable of being shared by multiple users 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.
- **For example:** The college Database organizes the data about the admin, staff, students and faculty etc. Using the database, you can easily retrieve, insert, and delete the information.

Database Management System (DBMS)

- Database management system is a software which allow users to create & maintain 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:

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.
- **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.

Database system

- The database & the associated DBMS software together are known as the **database system.**

Purpose of database systems

- **Reduction of redundancies:-**

  Centralized control of data by the Database administrator avoids unnecessary duplication of data and effectively reduces the total amount of data storage required. It also eliminates inconsistency and extra processing necessary to trace the data in large mass of data.
- **Shared data:-**

  A database allows the sharing of data under its control by any number of application programs or users.
- **Integrity:-**

  Centralized control can also ensure that adequate checks are incorporated in the DBMS to provide data integrity. Data integrity means that the data contained in the database is both accurate and consistent.
- **Security:-**

  Data is of vital importance. These should not be accessed by unauthorized persons. DBMS can ensure that proper access procedures are followed including proper authentication schemes for access to DBMS and additional checks before permitting access to sensitive data. Different levels of security could be implemented for various types of data and operations.
- **Conflict resolution:-**

  The conflicting requirements of various users and applications are solved by DBMS and best file structure and access methods are choosen to get optimal performance.
- **Data independence:-**

DBMS supports both physical and logical data independence. Data independence is advantageous in the database environment since it allows for changes at one level of the database without affection other levels.

Applications of database systems

Databases are widely used. Some of them are as follows.

- **Banking -** For customer information, accounts, loans and banking transactions.
- **Airlines -** For reservation and scheduled information.
- **Universities -** For student, course and grade information.
- **Credit card transactions -** For purposes on credit cards and generation of monthly statements.
- **Telecommunications -** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards and storing information about the communication network.
- **Finance -** For storing information about holdings, sales and purchases of financial instruments such as stocks and bonds.
- **Manufacturing -** For management of supply chain and for tracking production of items in factories, inventory of items in warehouses and orders for items.
- **Human resources -** For information about employees, salaries, payroll taxes& benefits and for generation of payments.

Elements of database

The four basic elements of database are:

1. **Data**

   Data is an individual unit that contains raw materials which do not carry any specific meaning. It can be used in a variety of forms like text, numbers, media, bytes, etc. it can be stored in pieces of paper or electronic memory, etc. **Data** doesn't depend on **information**.

2. **Relationship**

   Relationship is mapping association between data elements.

3. **Constraints**

   Constraints are used to maintain the accuracy or correctness of the data in the database.

4. **Schema**

   The description of a database is called database schema, which is specified during database design & is not expected to change frequently.

Features of database

- It should be shareable among different users and applications.
- It should be valid or correct with respect to the real world entity that they represent.
- It should be protected from unauthorized access & be secured.
- The consistency of the data should be maintained.
- It should be non-redundant. no two data item in a database should represent the same entity.
- Data should be independent of the applications, i.e. the DBA must be able to change the storage structure or access technique according to the changing requirements, without affecting the existing applications.

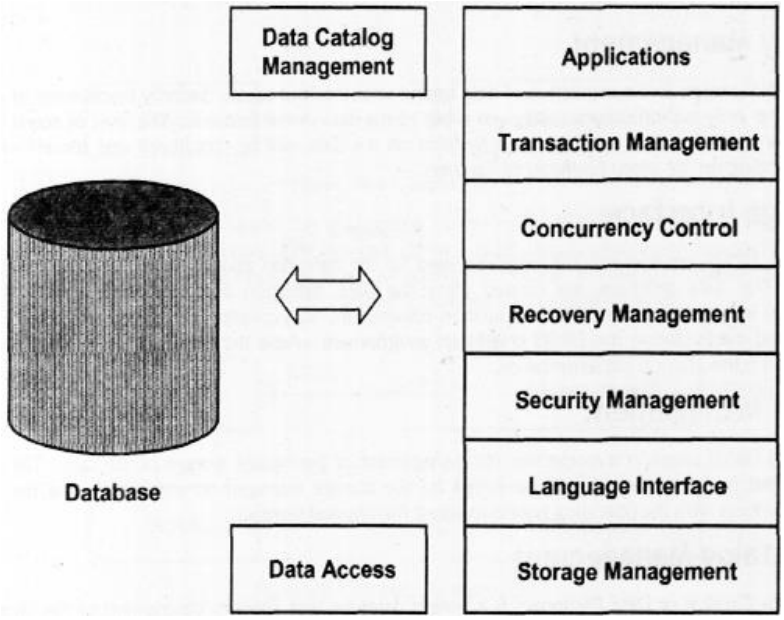
Characteristics of DBMS

- 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.

Components of DBMS

- A DBMS provides facilities for controlling data access, enforcing data integrity, managing concurrency control, and recovering the database after failures and restoring it from backup files, as well as maintaining database security.
- The DBMS performs the necessary operations on the stored database by using various components .
- The major components of a DBMS that integrate to manipulate the database and provide the services are shown in the following figure:

Services provided by a DBMS



Transaction Management

- A transaction is a sequence of database operations that represents a logical unit of work and that accesses a database and transforms it from one state to another.
- A transaction can update a record, delete one, modify a set of records, etc.
- When the DBMS does a 'commit' the changes made by the transaction are made permanent.
- If you don't want to make the changes permanent you can rollback the transaction and the database will remain in its original state.

Concurrency Control

- Concurrency control is the database management activity of coordinating the actions of database manipulation processes that operate concurrently that access shared data and can potentially interfere with one another.
- The goal of an ideal concurrency management mechanism is to allow concurrency while maintaining the consistency of the shared data.

Recovery Management

- The recovery management system in a database ensures that the aborted or failed transactions create no adverse effects on the database or the other transactions.
- Recovery mechanisms in a DBMS make sure that the database is returned to a consistent state after a transaction fails or aborts.

- Recovery is very much related to concurrency in the sense that, the mote the concurrency the more is the chance that an aborted transaction can affect many other transactions.

Security Management

- Security refers to the protection of data against unauthorized access.
- Security mechanisms of a DBMS make sure that only authorized users are given access to the data in the database.
- The level of access for each user and the operations that each user can perform on the data will be monitored and controlled by the DBMS depending on the access privileges of the users.

Language Interface

- The DBMS provides support languages used for the definition and manipulation of the data in the database.
- The data structures are created using the data definition language commands.
- The data manipulation is done using the data manipulation commands.
- By providing the language support for data definition and manipulation the DBMS creates an environment where the users can do their jobs without worrying about the physical implementation.

Storage Management

- The DBMS provides a mechanism for management of permanent storage of the data.
- The internal schema defines how the data should be stored by the storage management mechanism and the storage manager interfaces with the operating system to access the physical storage.

Data Catalog Management

- Data Catalog or Data Dictionary is a system database that contains descriptions of the data in the database (metadata).
- It contains information about data, relationships, constraints and the entire schema that organize these features into a unified database.
- The data catalog can be queried to get information about the structure of the database.

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**.

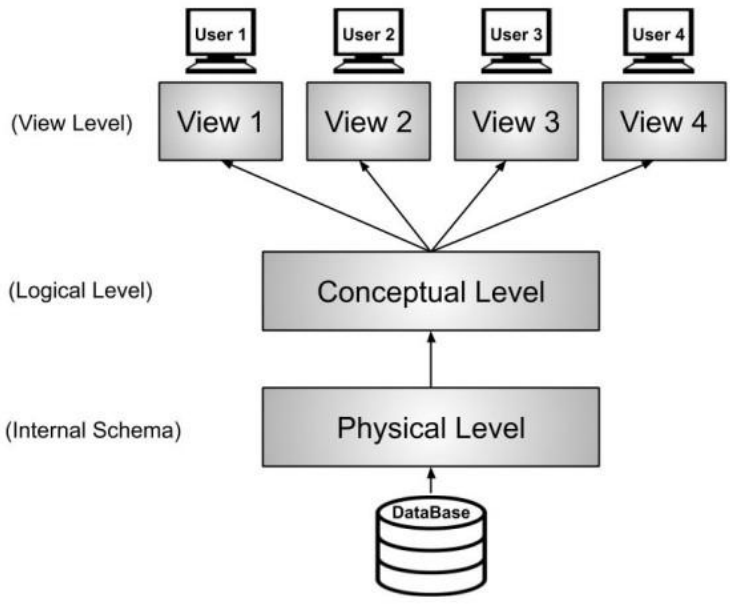
Example: If we want to access any mail from our Gmail then we don't know where that data is physically stored i.e is the data present in India or USA or what data model has been used to store that data? We are not concerned about these things. We are only concerned with our email. So, information like these i.e. location of data and data models are irrelevant to us and in data abstraction, we do this only. Apart from the location of data and data models, there are other factors that we don't care of. We hide the unnecessary data from the user and this process of hiding unwanted data is called Data Abstraction.

There are mainly three levels of data abstraction and we divide it into three levels in order to achieve *Data Independence*.

Data Independence means users and data should not directly interact with each other. The user should be at a different level and the data should be present at some other level. By doing so, Data Independence can be achieved.

So, let's see in details what are these three levels of data abstraction:

1. View Level
2. Conceptual Level
3. Physical Level



Levels of Data Abstraction

1. **View level**: Highest level of data abstraction. This level describes the user interaction with database system.

2. **Logical level**: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

3. **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.

Example: Let's say we are storing customer information in a customer table. 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.

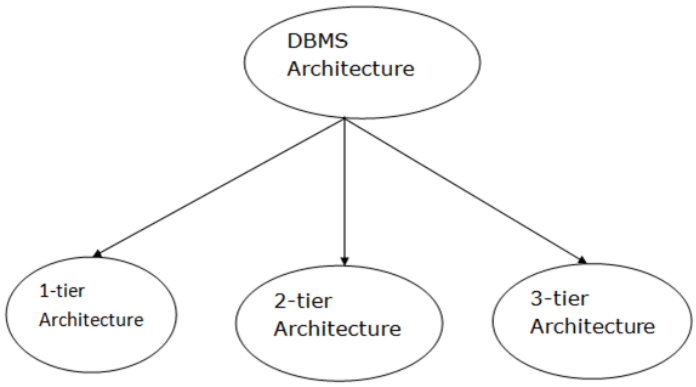
At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

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.

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

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

2-Tier Architecture

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

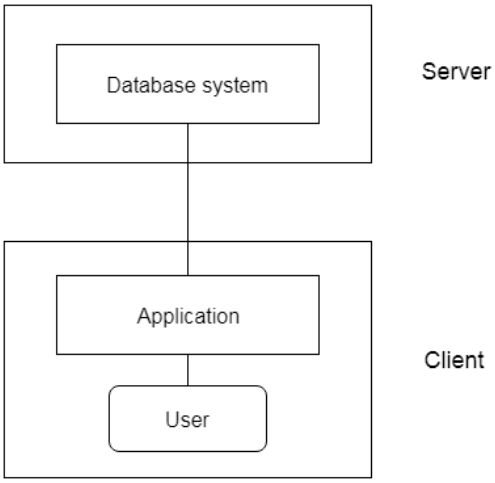


Fig: 2-tier Architecture

3-Tier Architecture

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

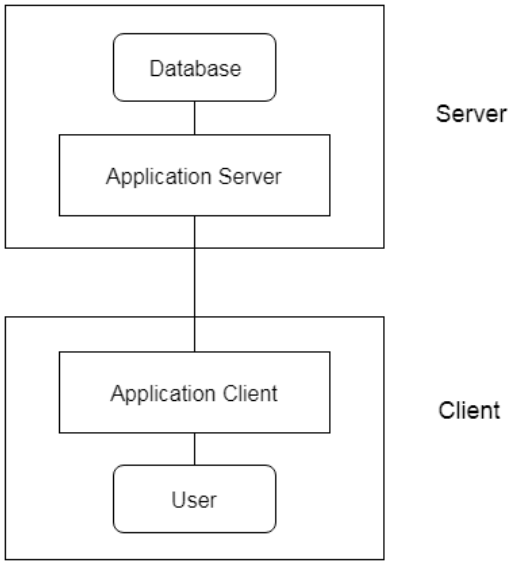
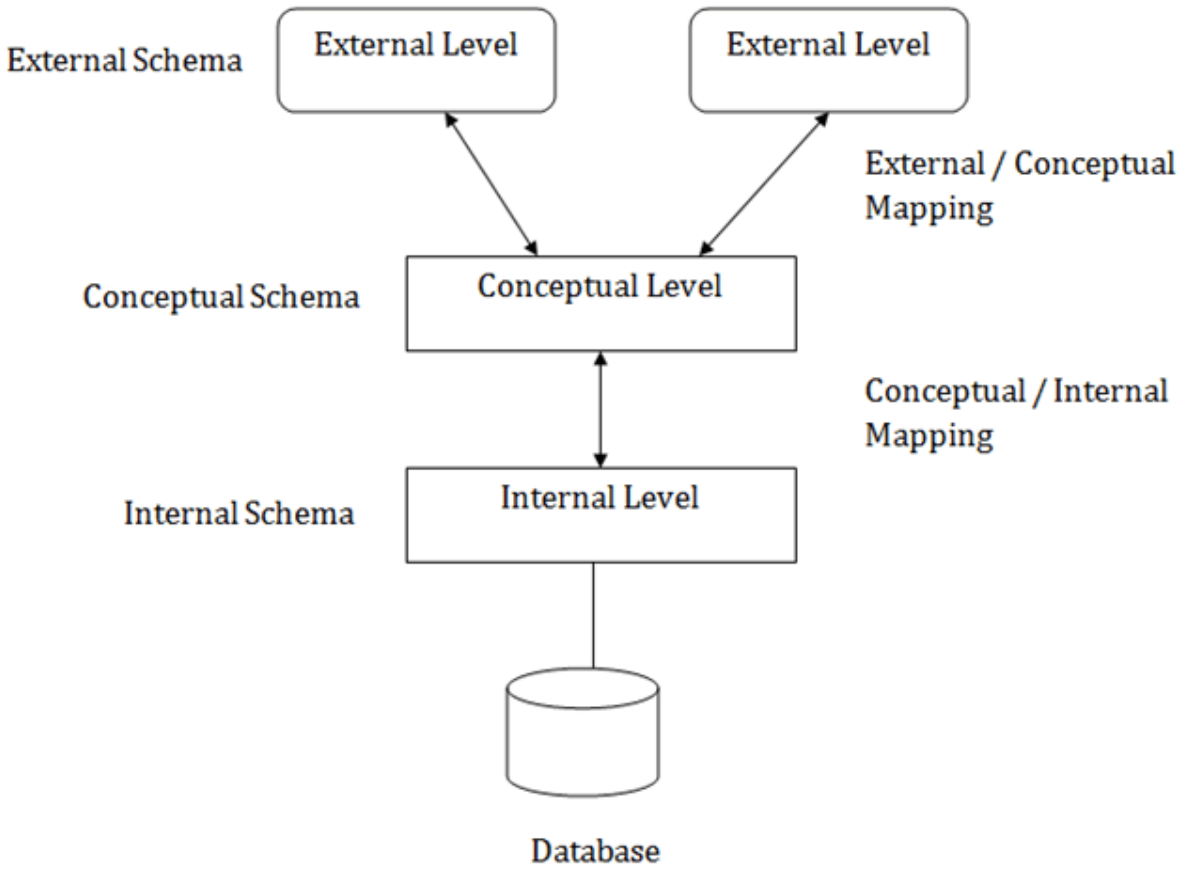


Fig: 3-tier Architecture

Three schema Architecture

- The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
- This framework is used to describe the structure of a specific database system.
- The three schema architecture is also used to separate the user applications and physical database.
- The three schema architecture contains three-levels. It breaks the database down into three different categories.

The three-schema architecture is as follows:



In the above diagram:

- It shows the DBMS architecture.

- Mapping is used to transform the request and response between various database levels of architecture.
- Mapping is not good for small DBMS because it takes more time.
- In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema.
- In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.

1. Internal Level

- The internal level has an internal schema which describes the physical storage structure of the database.
- The internal schema is also known as a physical schema.
- It uses the physical data model. It is used to define that how the data will be stored in a block.
- The physical level is used to describe complex low-level data structures in detail.

2. Conceptual Level

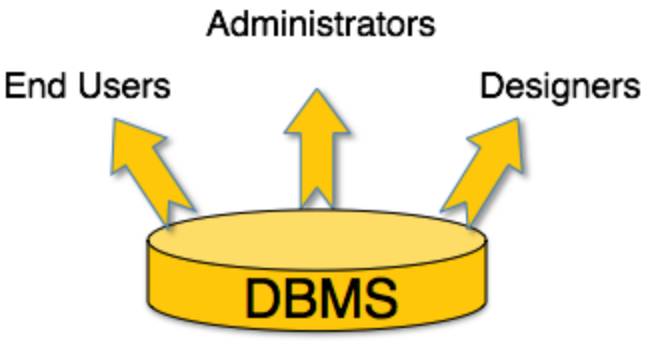
- The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
- The conceptual schema describes the structure of the whole database.
- The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
- In the conceptual level, internal details such as an implementation of the data structure are hidden.
- Programmers and database administrators work at this level.

3. External Level

- At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
- An external schema is also known as view schema.
- Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
- The view schema describes the end user interaction with database systems.

Database Users

- A typical DBMS has users with different rights and permissions who use it for different purposes.
- Some users retrieve data and some back it up.
- The users of a DBMS can be broadly categorized as follows −



1. **Administrators** –
    - Administrators maintain the DBMS and are responsible for administrating the database.
    - They are responsible to look after its usage and by whom it should be used.
    - They create access profiles for users and apply limitations to maintain isolation and force security.
    - Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.
2. **Designers** –
    - Designers are the group of people who actually work on the designing part of the database.
    - They keep a close watch on what data should be kept and in what format.
    - They identify and design the whole set of entities, relations, constraints, and views.
3. **End Users** –
    - End users are those who actually reap the benefits of having a DBMS.
    - End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.
    - **Casual end users:-**

Casual end user occasionally access the database , but they may need different information each time.

- **Naïve or parametric end users :-**

  Naïve or parametric end users make up a sizeable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates, called canned transaction, that have been carefully programmed and tested.

- **Sophisticated end users :-**

  Sophisticated end users include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.

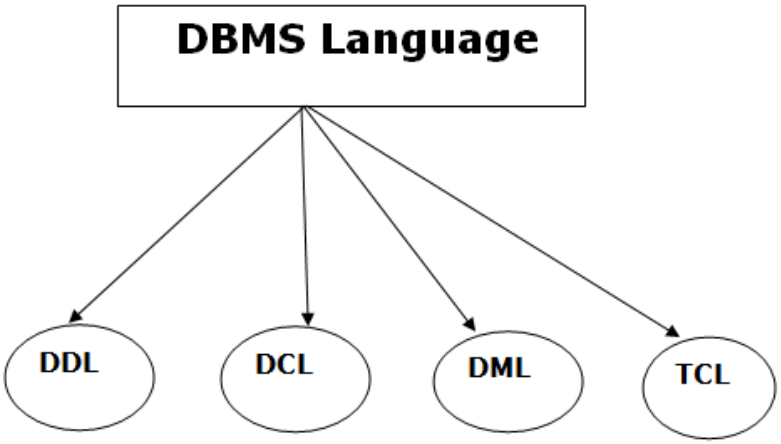
- **Stand-alone end users :-**

  Stand alone users maintain personal databases by using readymade program packages that provide easy to use menu or graphic based interfaces.

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.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

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.

Data Dictionary

A data dictionary contains metadata i.e data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary, it is only handled by the database administrators.

The data dictionary in general contains information about the following −

- Names of all the database tables and their schemas.

- Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
- Physical information about the tables such as where they are stored and how.
- Table constraints such as primary key attributes, foreign key information etc.
- Information about the database views that are visible.

Advantages of DBMS

- **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time:** It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- **multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.
- **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.

DBMS vs. File System

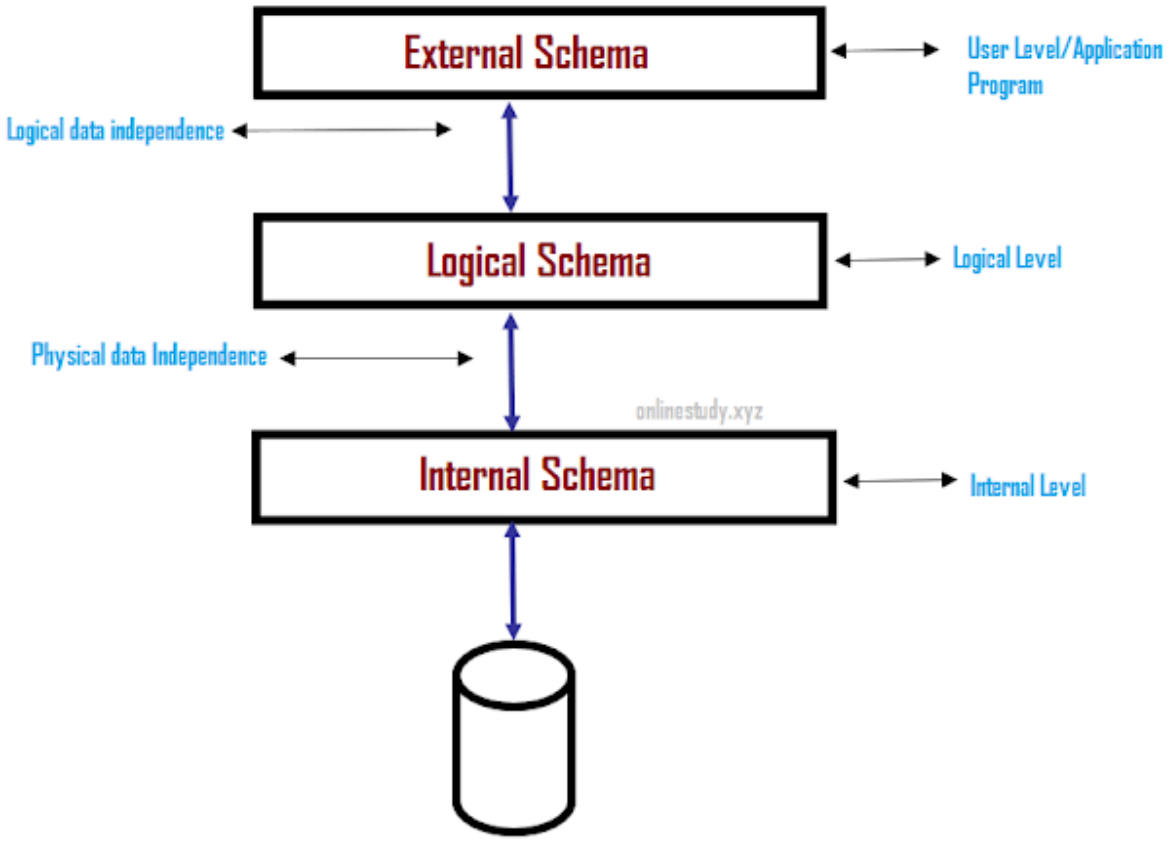
There are following differences between DBMS and File system:

DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	File system is a collection of data. In this system, the user has to write the procedures for managing the database.
DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.
DBMS provides a good protection mechanism.	It is very difficult to protect a file under the file system.
DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.
DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

UNIT – 2 DATA MODELS

Data Independence

- Data independence can be explained using the three-schema architecture.
- Data Independence is defined as a property of DBMS that helps you to change the Database schema at one level of a database system without requiring to change the schema at the next higher level.
- Data independence helps you to keep data separated from all programs that make use of it.
- There are two types of data independence:



Importance of Data Independence

- Helps you to improve the quality of the data
- Database system maintenance becomes affordable
- Enforcement of standards and improvement in database security
- You don't need to alter data structure in application programs
- Permit developers to focus on the general structure of the Database rather than worrying about the internal implementation
- It allows you to improve state which is undamaged or undivided
- Database incongruity is vastly reduced.
- Easily make modifications in the physical level is needed to improve the performance of the system.

1. Physical Data Independence :

- Physical Data Independence is defined as the ability to make changes in the structure of the lowest level of the Database Management System (DBMS) without affecting the higher-level schemas.
- Hence, modification in the Physical level should not result in any changes in the Logical or View levels.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.
- Physical Data Independence is achieved by modifying the physical layer to logical layer mapping (PL-LL mapping). We must ensure that the modification we have done is localized.

Examples of changes under Physical Data Independence

Due to Physical independence, any of the below change will not affect the conceptual layer.

- Using a new storage device like Hard Drive or Magnetic Tapes
- Modifying the file organization technique in the Database
- Switching to different data structures.
- Changing the access method.

- Modifying indexes.
- Changes to compression techniques or hashing algorithms.
- Change of Location of Database from say C drive to D Drive

2. Logical Data Independence :

- Logical Data Independence is defined as the ability to make changes in the structure of the middle level of the Database Management System (DBMS) without affecting the highest-level schema or application programs.
- Hence, modification in the logical level should not result in any changes in the view levels or application programs.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.
- Logical Data Independence is achieved by modifying the view layer to logical layer mapping (VL-LL mapping).

Examples of changes under Logical Data Independence

Due to Logical independence, any of the below change will not affect the external layer.

- Add/Modify/Delete a new attribute, entity or relationship is possible without a rewrite of existing application programs
- Merging two records into one
- Breaking an existing record into two or more records

Data Model

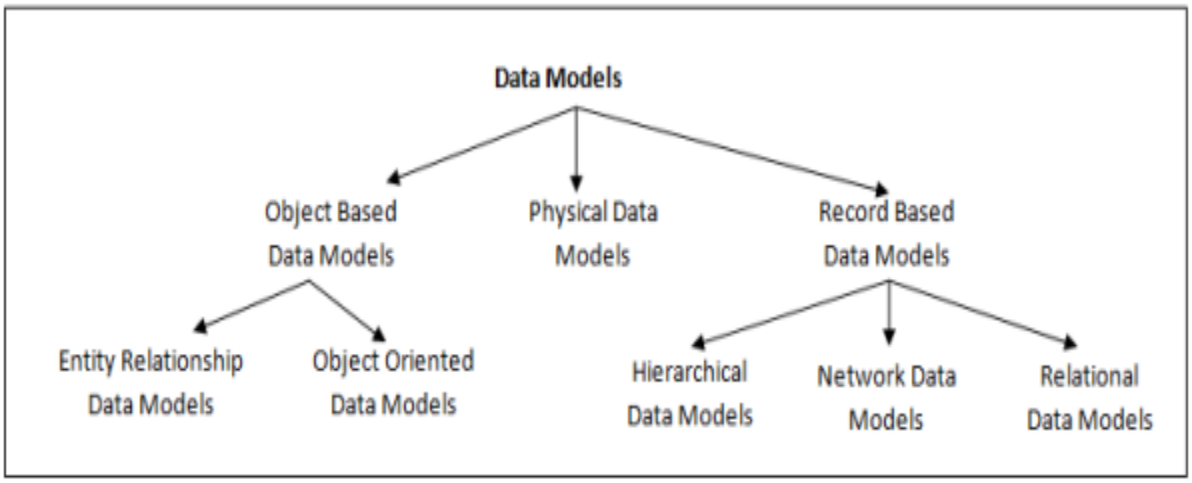
- The **Data Model** is defined as an abstract model that organizes data description, data semantics, and consistency constraints of data.
- The data model emphasizes on what data is needed and how it should be organized instead of what operations will be performed on data.
- Data Model is like an architect's building plan, which helps to build conceptual models and set a relationship between data items.

Why use Data Model?

The primary goal of using data model are:

- Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.
- A data model helps design the database at the conceptual, physical and logical levels.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.
- It provides a clear picture of the base data and can be used by database developers to create a physical database.
- It is also helpful to identify missing and redundant data.
- Though the initial creation of data model is labor and time consuming, in the long run, it makes your IT infrastructure upgrade and maintenance cheaper and faster.

Types of Data Model



Object based Data Models

- In object based data models, the focus is on how data is represented.
- The data is divided into multiple entities each of which have some defining characteristics.
- Moreover, these data entities are connected with each other through some relationships.

- So, in object based data models the entities are based on real world models, and how the data is in real life.
- There is not as much concern over what the data is as compared to how it is visualised and connected.
- It describe data at the conceptual and view levels.
    - E-R Model
    - Object oriented Model

Entity-Relationship Model

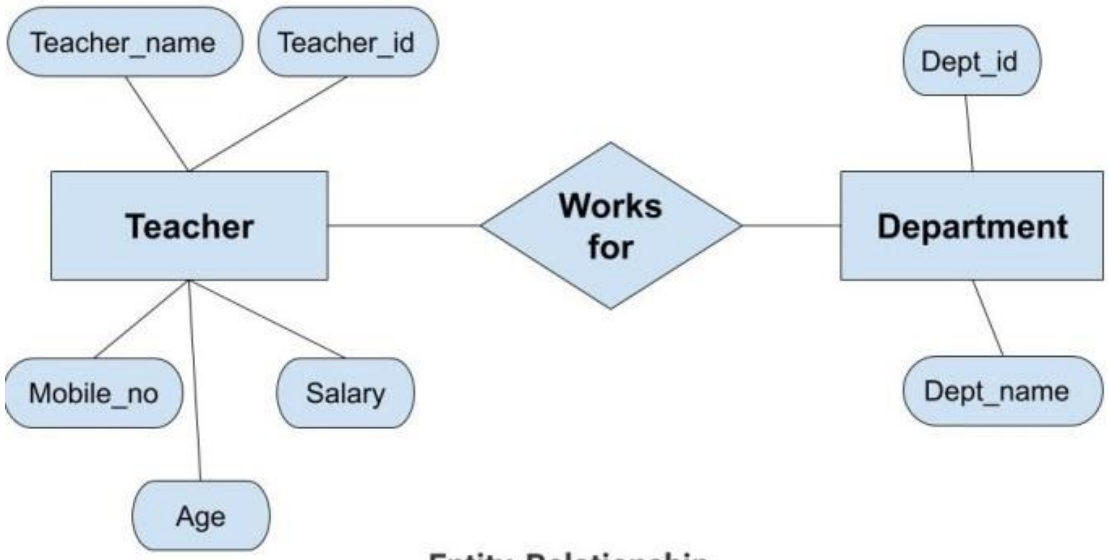
- Entity-Relationship Model or simply ER Model is a high-level data model diagram.
- In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand.
- It is also very easy for the developers to understand the system by just looking at the ER diagram.
- We use the ER diagram as a visual tool to represent an ER Model.
- ER diagram has the following three components:
    - ***Entities:*** Entity is a real-world thing. It can be a person, place, or even a concept.

      Example: Teachers, Students, Course, Building, Department, etc are some of the entities of a School Management System.
    - ***Attributes:*** An entity contains a real-world property called attribute. This is the characteristics of that attribute.

      Example: The entity teacher has the property like teacher id, salary, age, etc.
    - ***Relationship:*** Relationship tells how two attributes are related.

      Example: Teacher works for a department.

Example:



Entity-Relationship Model

- In the above diagram, the entities are Teacher and Department.
- The attributes of ***Teacher*** entity are Teacher_Name, Teacher_id, Age, Salary, Mobile_Number.
- The attributes of entity ***Department*** entity are Dept_id, Dept_name.
- The two entities are connected using the relationship.
- Here, each teacher works for a department.

Features of ER Model

1. **Graphical Representation for Better Understanding:** It is very easy and simple to understand so it can be used by the developers to communicate with the stakeholders.
2. **ER Diagram:** ER diagram is used as a visual tool for representing the model.
3. **Database Design:** This model helps the database designers to build the database and is widely used in database design.

Advantages of ER Model

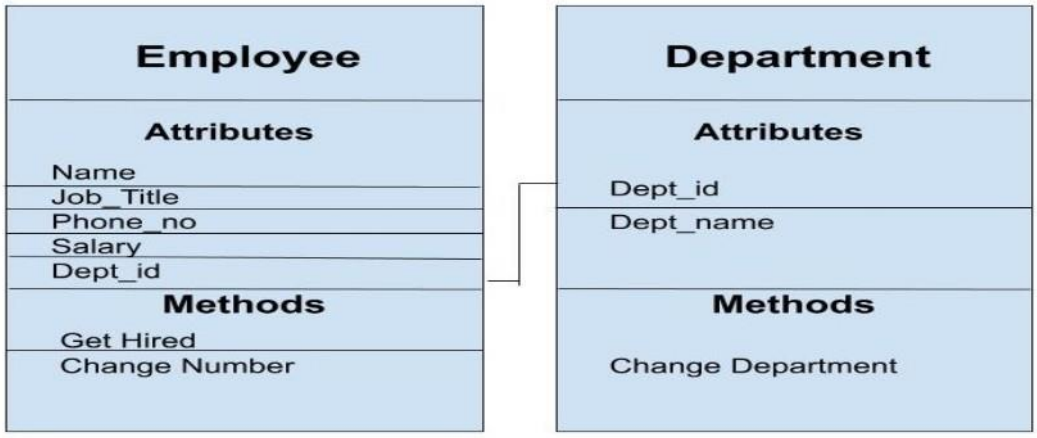
- **Simple:** Conceptually ER Model is very easy to build. If we know the relationship between the attributes and the entities we can easily build the ER Diagram for the model.
- **Effective Communication Tool**: This model is used widely by the database designers for communicating their ideas.
- **Easy Conversion to any Model**: This model maps well to the relational model and can be easily converted relational model by converting the ER model to the table. This model can also be converted to any other model like network model, hierarchical model etc.

Disadvantages of ER Model

- **No industry standard for notation:** There is no industry standard for developing an ER model. So one developer might use notations which are not understood by other developers.
- **Hidden information:** Some information might be lost or hidden in the ER model. As it is a high-level view so there are chances that some details of information might be hidden.

Object-Oriented Data Model

- The real-world problems are more closely represented through the object-oriented data model.
- In this model, both the data and relationship are present in a single structure known as an object.
- We can store audio, video, images, etc in the database which was not possible in the relational model
- In this model, two are more objects are connected through links.
- We use this link to relate one object to other objects. This can be understood by the example given below.



Object_Oriented_Model

- In the above example, we have two objects Employee and Department.
- All the data and relationships of each object are contained as a single unit.
- The attributes like Name, Job_title of the employee and the methods which will be performed by that object are stored as a single object.
- The two objects are connected through a common attribute i.e the Department_id and the communication between these two will be done with the help of this common id.

Physical Data Model

- A Physical Data Model describes a database-specific implementation of the data model.
- Ultimately, all data in a database is stored physically on a secondary storage device such as discs and tapes.
- This is stored in the form of files, records and certain other data structures.
- It has all the information of the format in which the files are present and the structure of the databases, presence of external data structures and their relation to each other.

Characteristics of a physical data model:

- The physical data model describes data need for a single project or application though it may be integrated with other physical data models based on project scope.
- Data Model contains relationships between tables that which addresses cardinality and nullability of the relationships.
- Columns should have exact datatypes, lengths assigned and default values.

- Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

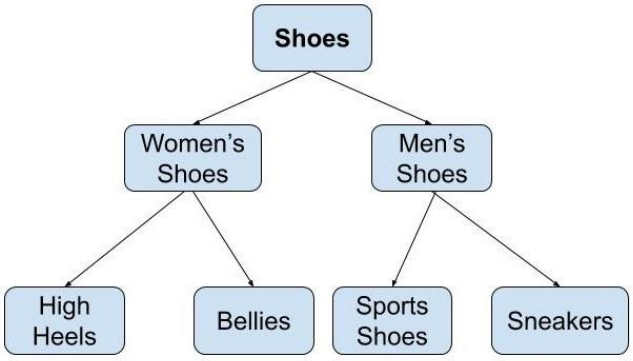
Record Based Data Models

- Like Object based model, they also describe data at the conceptual and view levels.
- These models specify logical structure of database with records, fields and attributes.

    - Hierarchical Model
    - Network Model
    - Relational Model

Hierarchical Model

- Hierarchical Model was the first DBMS model.
- This model organises the data in the hierarchical tree structure.
- The hierarchy starts from the root which has root data and then it expands in the form of a tree adding child node to the parent node.
- This model easily represents some of the real-world relationships like food recipes, sitemap of a website etc.
- ***Example:*** We can represent the relationship between the shoes present on a shopping website in the following way:



Hierarchical Model

Features of a Hierarchical Model

1. **One-to-many relationship:** The data here is organized in a tree-like structure where the one-to-many relationship is between the data types. Also, there can be only one path from parent to any node.

   Example: In the above example, if we want to go to the node sneakers we only have one path to reach there i.e through men's shoes node.

2. **Parent-Child Relationship:** Each child node has a parent node but a parent node can have more than one child node. Multiple parents are not allowed.
3. **Deletion Problem:** If a parent node is deleted then the child node is automatically deleted.
4. **Pointers:** Pointers are used to link the parent node with the child node and are used to navigate between the stored data.

   Example: In the above example the 'shoes' node points to the two other nodes 'women shoes' node and 'men's shoes' node.

Advantages of Hierarchical Model

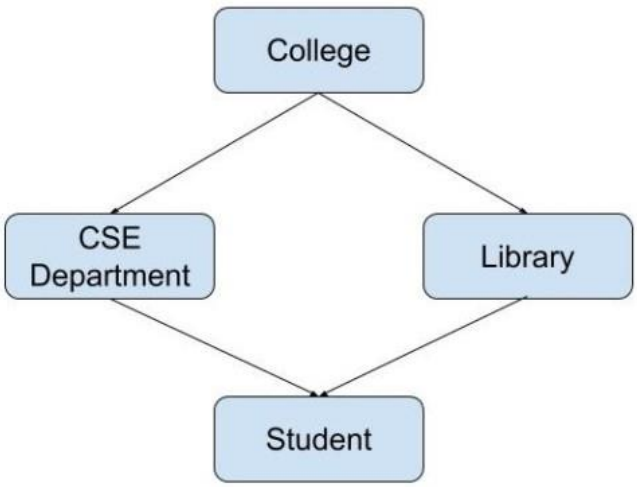
- It is very simple and fast to traverse through a tree-like structure.
- Any change in the parent node is automatically reflected in the child node so, the integrity of data is maintained.

Disadvantages of Hierarchical Model

- Complex relationships are not supported.
- As it does not support more than one parent of the child node so if we have some complex relationship where a child node needs to have two parent node then that can't be represented using this model.
- If a parent node is deleted then the child node is automatically deleted.

Network Model

- This model is an extension of the hierarchical model.
- It was the most popular model before the relational model.
- This model is the same as the hierarchical model, the only difference is that a record can have more than one parent.
- It replaces the hierarchical tree with a graph.
- ***Example:*** In the example below we can see that node student has two parents i.e. CSE Department and Library. This was earlier not possible in the hierarchical model.



Network Model

Features of a Network Model

1. **Ability to Merge more Relationships:** In this model, as there are more relationships so data is more related. This model has the ability to manage one-to-one relationships as well as many-to-many relationships.
2. **Many paths:** As there are more relationships so there can be more than one path to the same record. This makes data access fast and simple.
3. **Circular Linked List:** The operations on the network model are done with the help of the circular linked list. The current position is maintained with the help of a program and this position navigates through the records according to the relationship.

Advantages of Network Model

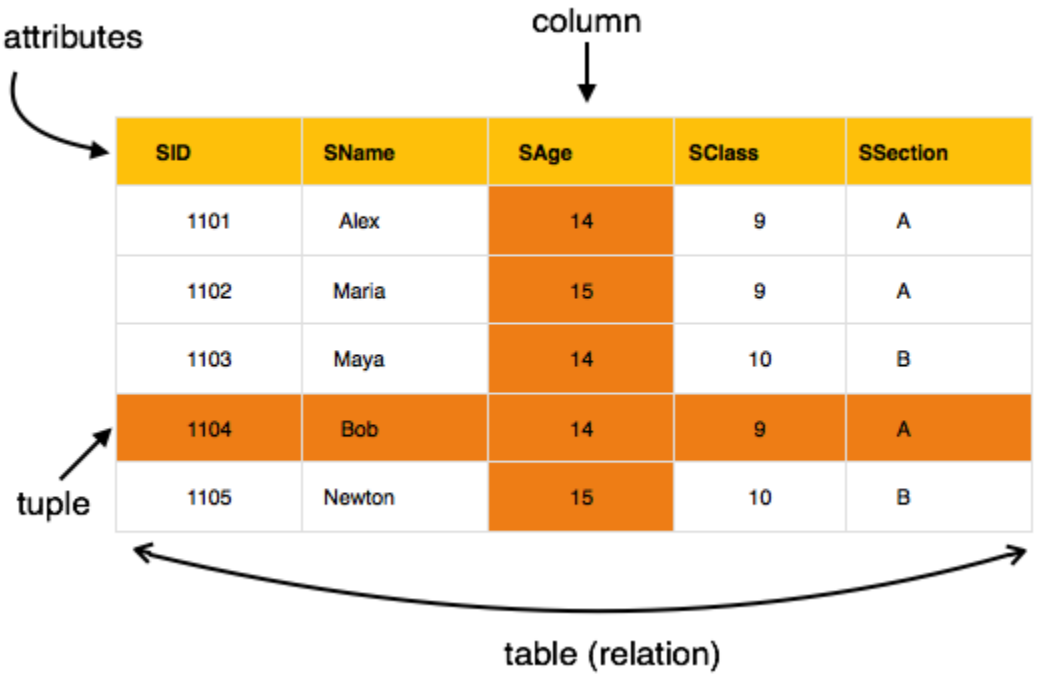
- The data can be accessed faster as compared to the hierarchical model. This is because the data is more related in the network model and there can be more than one path to reach a particular node. So the data can be accessed in many ways.
- As there is a parent-child relationship so data integrity is present. Any change in parent record is reflected in the child record.

Disadvantages of Network Model

- As more and more relationships need to be handled the system might get complex. So, a user must be having detailed knowledge of the model to work with the model.
- Any change like updation, deletion, insertion is very complex.

Relational Model

- Relational Model is the most widely used model.
- In this model, the data is maintained in the form of a two-dimensional table.
- All the information is stored in the form of row and columns.
- The basic structure of a relational model is tables.
- So, the tables are also called relations in the relational model.



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

Features of Relational Model

1. **Tuples**: Each row in the table is called tuple. A row contains all the information about any instance of the object. In the above example, each row has all the information about any specific individual like the first row has information about Alex.
2. **Attribute or field:** Attributes are the property which defines the table or relation. The values of the attribute should be from the same domain. In the above example, we have different attributes of the student like SAge, SClass etc.

Advantages of Relational Model

- **Simple:** This model is simpler as compared to the network and hierarchical model.
- **Scalable:** This model can be easily scaled as we can add as many rows and columns we want.
- **Structural Independence:** We can make changes in database structure without changing the way to access the data. When we can make changes to the database structure without affecting the capability to DBMS to access the data we can say that structural independence has been achieved.

Disadvantages of Relational Model

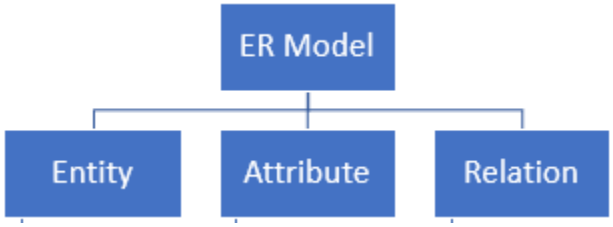
- **Hardware Overheads:** For hiding the complexities and making things easier for the user this model requires more powerful hardware computers and data storage devices.
- **Bad Design:** As the relational model is very easy to design and use. So the users don't need to know how the data is stored in order to access it. This ease of design can lead to the development of a poor database which would slow down if the database grows.

E-R Model

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

- **E-R Model** is a high-level conceptual data model diagram.
- ER model helps to systematically analyze data requirements to produce a well-designed database.
- The ER Model represents real-world entities and the relationships between them.

COMPONENTS OF E-R MODEL



ENTITY

- An entity is a real-world thing either living or non-living that is easily recognizable and non-recognizable.
- 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.

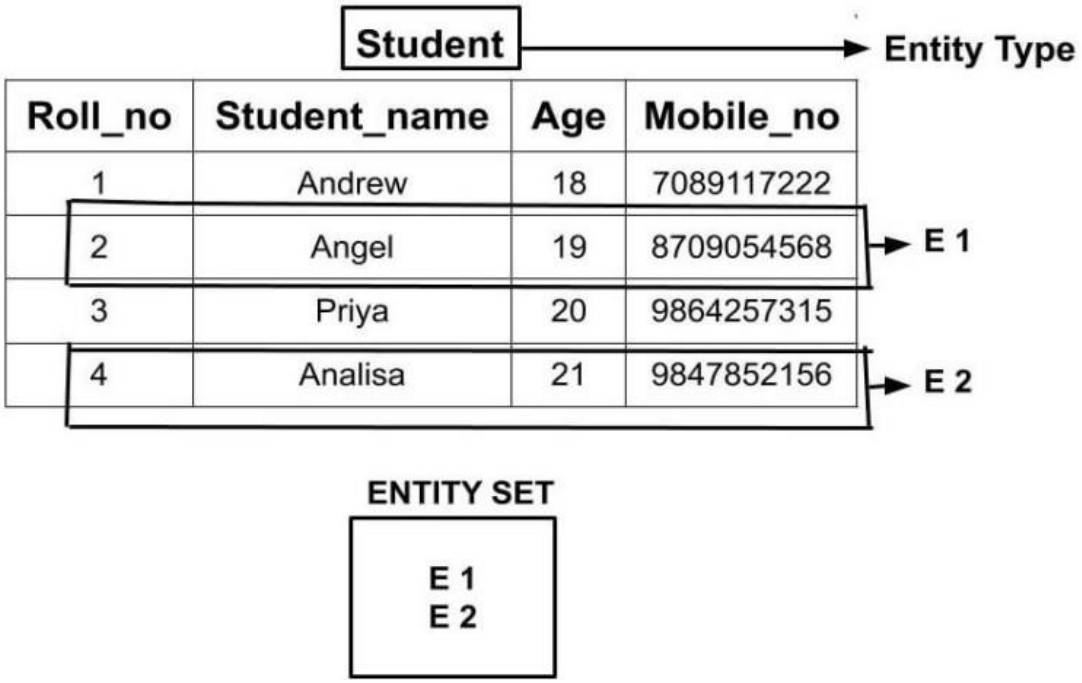
Examples of entities:

- **Person:** Employee, Student, Patient
- **Place:** Store, Building
- **Object:** Machine, product, and Car
- **Event:** Sale, Registration, Renewal
- **Concept:** Account, Course

Entity Type, Entity Set –

- An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

- An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



ATTRIBUTES

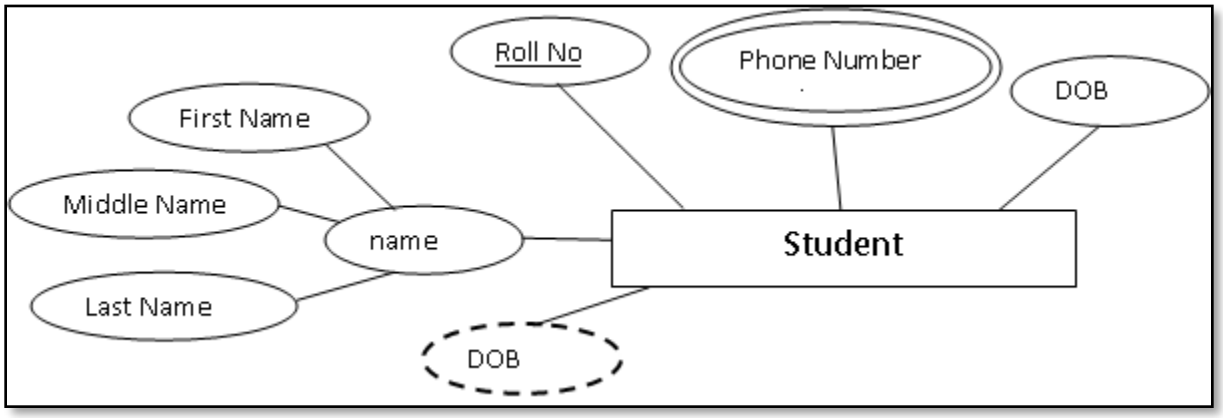
- Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.
- There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- **Simple attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For

example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

- **Single-value attribute** − Single-value attributes contain single value. For example − Social_Security_Number.
- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

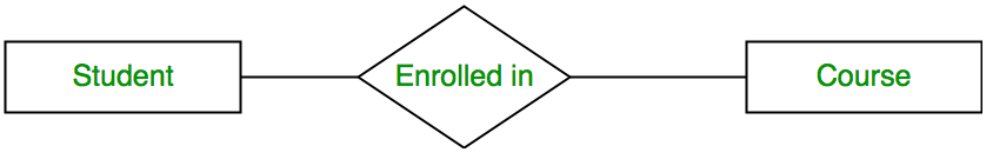


Relationship

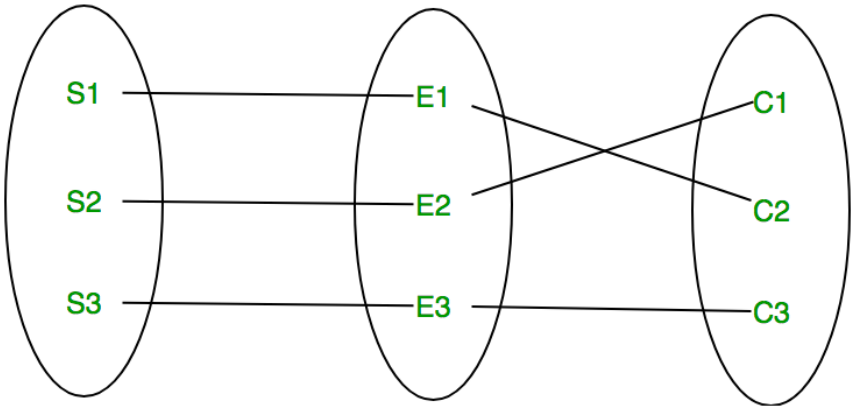
- The association among entities is called a relationship.
- Relationship are represented by diamond shaped symbols.
- For example, an employee **works_at** a department, a student **enrolls** in a course. Here, **Works_at** and **Enrolls** are called relationships.

Relationship Type and Relationship Set:

- 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.



- 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.



MAPPING CONSTRAINTS

Some terms associated with entities and relationships. These are

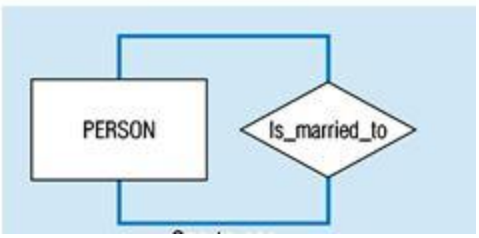
- Degree
- Connectivity
- Cardinality
- Dependency
- Participation

DEGREE

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

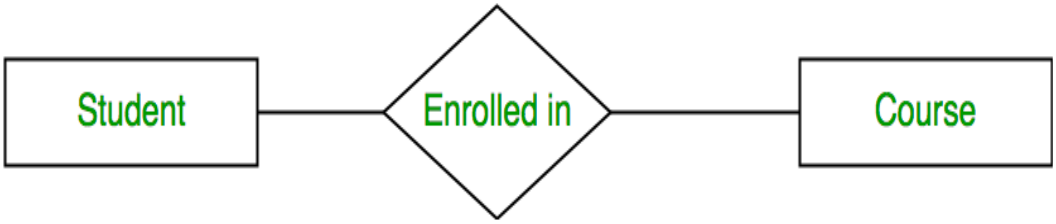
1. **Unary Relationship –**

   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.



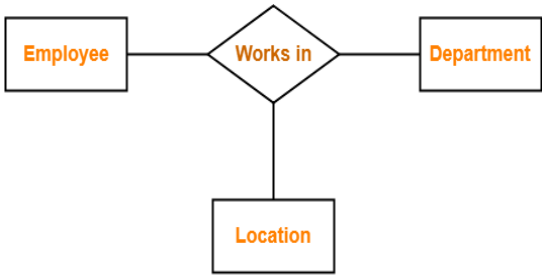
2. Binary Relationship –

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

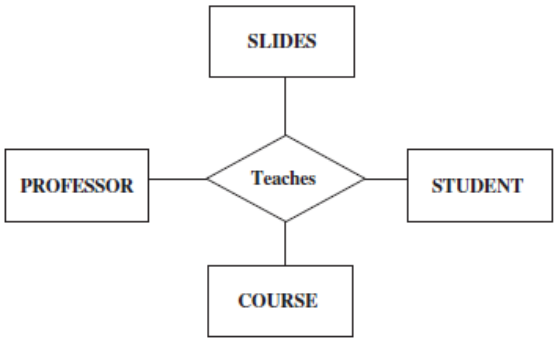


3. n-ary Relationship –

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

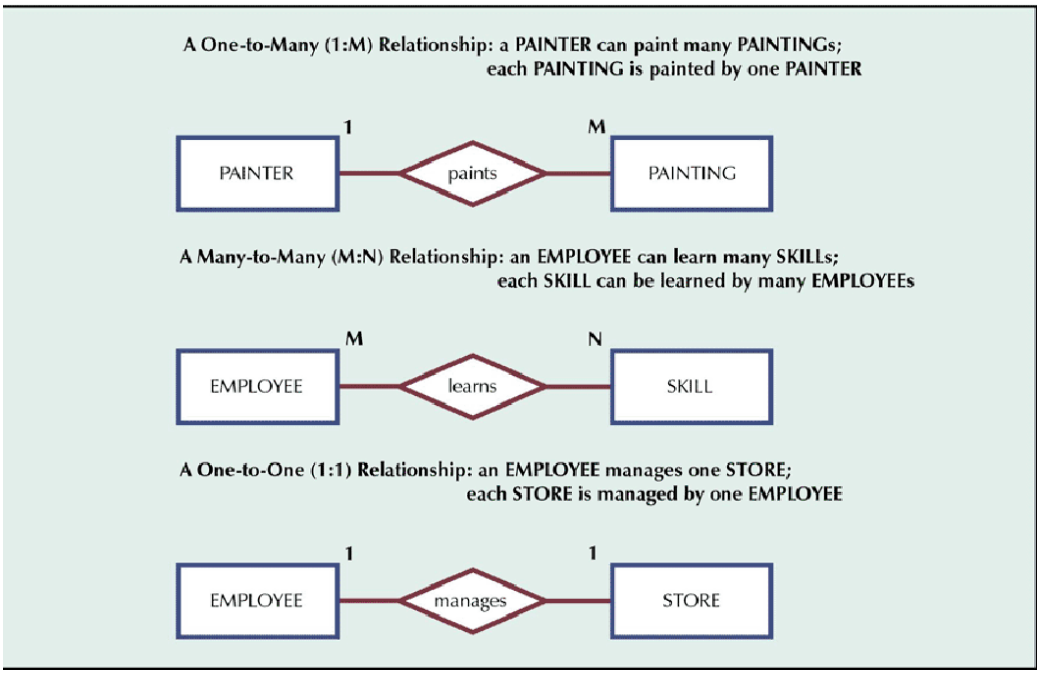


Ternary Relationship Set



CONNECTIVITY

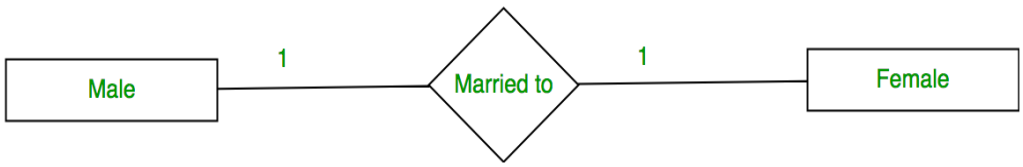
- The term connectivity is used to describe the relationship classification. The E-R diagram indicates the relationship's connectivity by placing a 1, M or N near the related entities.



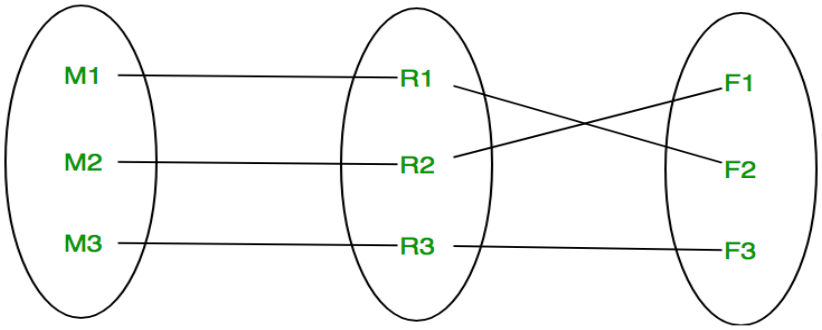
CARDINALITY

- 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:

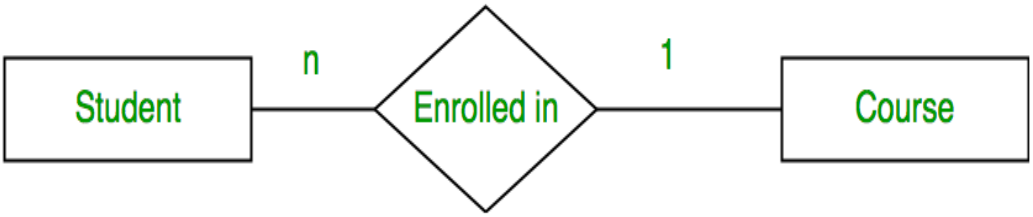
1. **One to one –**
    - When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one.
    - Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.



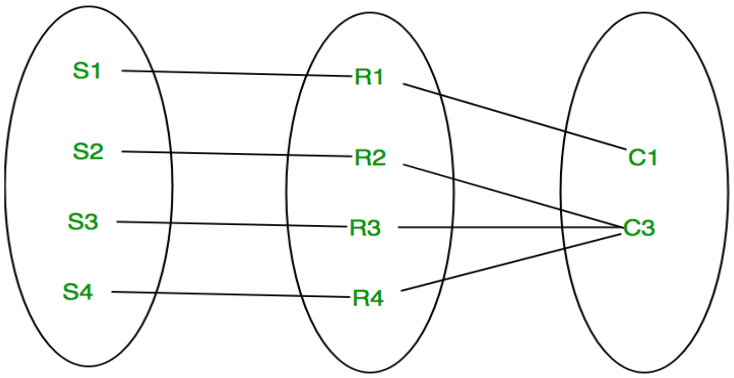
Using Sets, it can be represented as:



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.
   - Let us assume that 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.



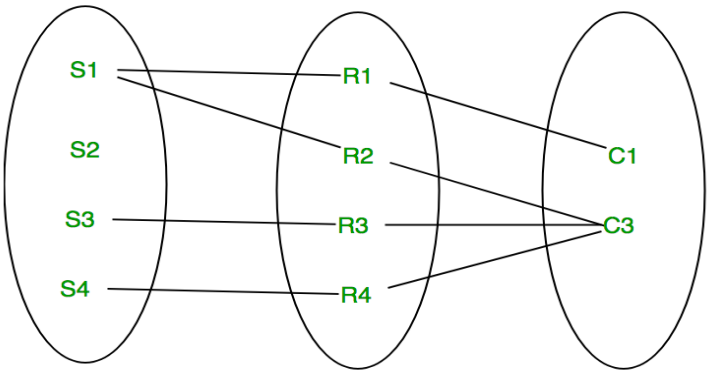
Using Sets, it can be represented as:



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.
    - Let us assume that 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.

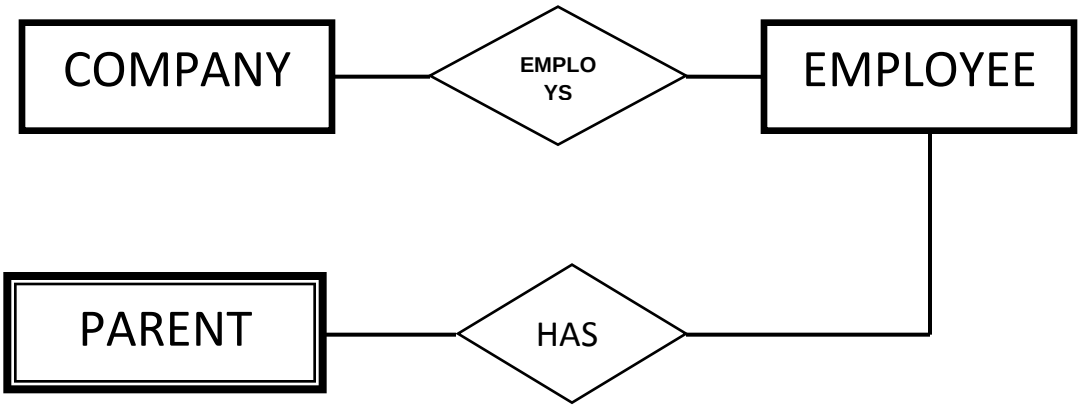
Using sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

DEPENDENCY

- Entities are classified as strong and weak entity types or existence dependent.
- Existence dependency means that if the existence of entity x depends on the existence of entity y, the x is said to be existence dependent on y.
- Operationally if y is deleted , so is x
- Entity y is said to be a dominant entity & x is said to be a subordinate entity.
- A strong entity is one which is not existence dependent.
- A weak entity is dependent on the existence of another entity.
- Weak entity also referred to as child dependent or subordinate entities & strong entities as parent, owner or dominant entities.



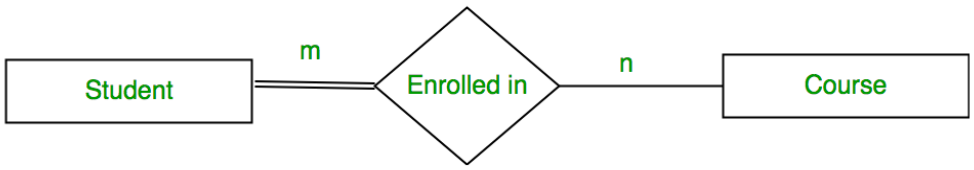
WEAK & STRONG ENTITY

- In the above diagram PARENT is a weak entity, as it needs the entity EMPLOYEE for its existence.
- The Entity COMPANY & EMPLOYEE are strong entities.
- Weak entity is representing in double rectangular symbol.

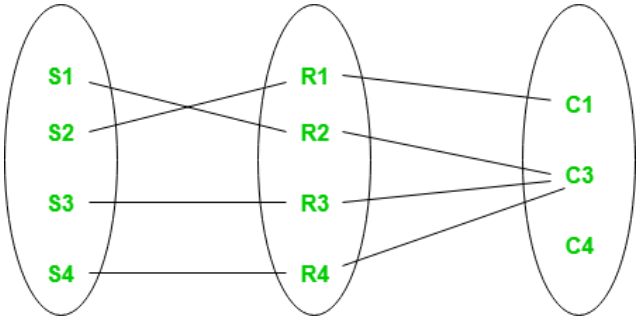
PARTICIPATION

- 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 participat**e 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,



Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

ER Diagram

- **ER Diagram** stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database.
- In other words, ER diagrams help to explain the logical structure of databases.
- ER diagrams are created based on three basic concepts: entities, attributes and relationships.
- ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.
- At first look, an ER diagram looks very similar to the flowchart.
- However, ER Diagram includes many specialized symbols, and its meanings make this model unique.
- The purpose of ER Diagram is to represent the entity framework infrastructure.

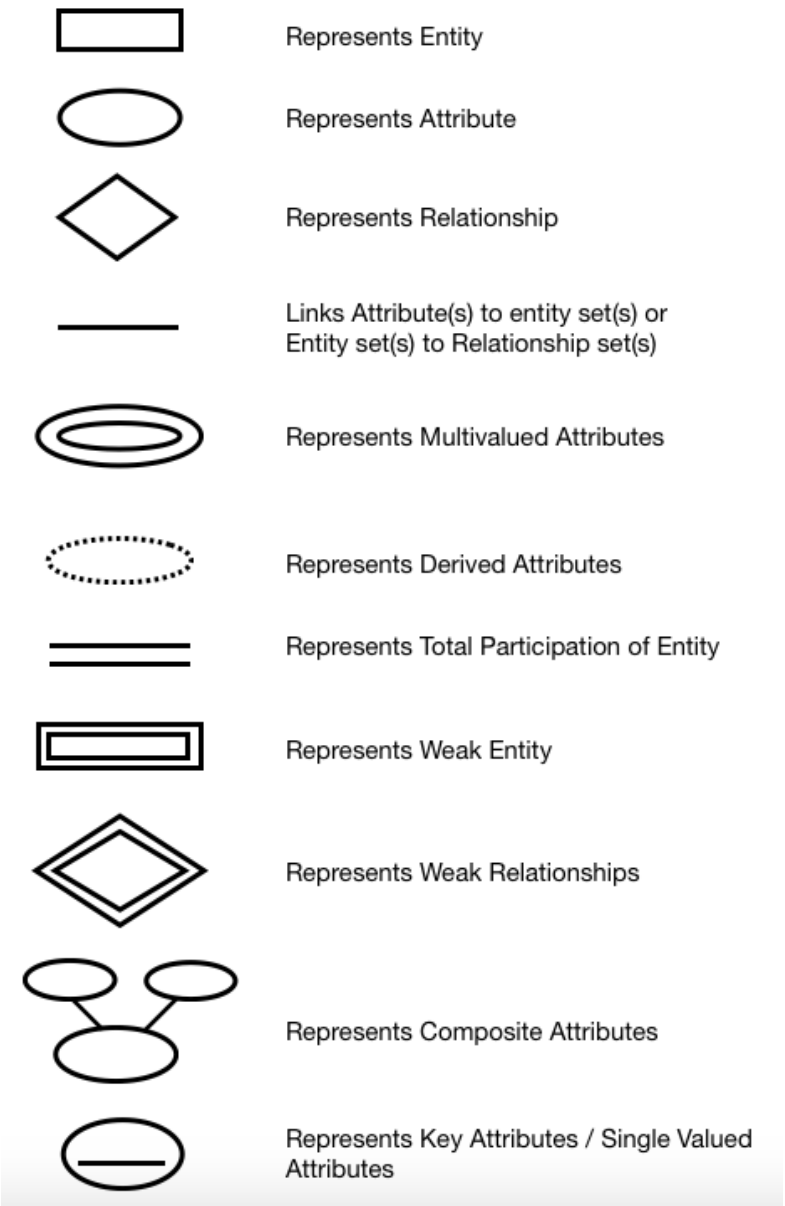
Why use ER Diagrams?

- 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 allows 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 ERP diagram
- ERD Diagram allows you to communicate with the logical structure of the database to users

Different notations used for ER Diagram in DBMS

- Below picture clearly depicts about different notations we need while using ER diagram in DBMS. They are –

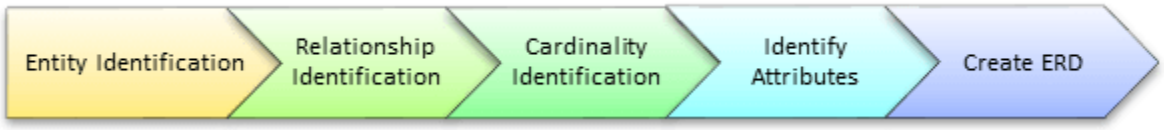


As above image describes,

- **Rectangle** – It represents entity in the ER Model.
- **Ellipse** – It represents attribute in the ER Model.
- **Diamond** – It represents relationship between entity and attribute.
- **Line** – It links attribute(s) to entity set(s) and entity set(s) to relationship set(s).
- **Doubles Ellipses** – It represents multivalued attributes.
- **Dashed Ellipses** – It denotes derived attributes.
- **Double lines** – It indicates total participation of an entity in a relationship set.
- **Double Rectangle** – It represents weak entity set.
- **Double Diamonds** – It represents weak relationships.
- **Multiple ellipses connected to single ellipse using lines** – It represents composite attribute
- **Ellipse with line inside it** – It represents single values attributes

How to Create an Entity Relationship Diagram (ERD)

Following are the steps to create an ER Diagram:



Steps to Create an ER Diagram

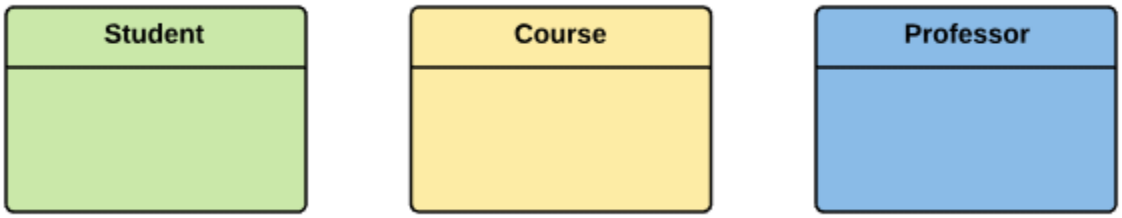
Example:

```
In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course
```

Step 1) Entity Identification

We have three entities

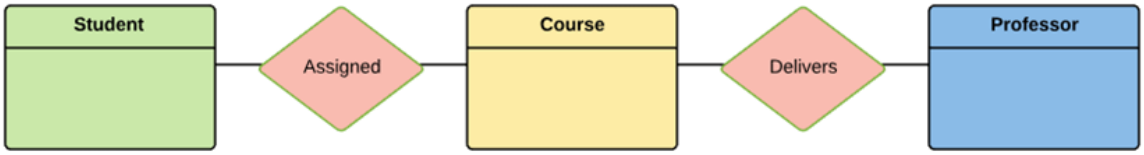
- Student
- Course
- Professor



Step 2) Relationship Identification

We have the following two relationships

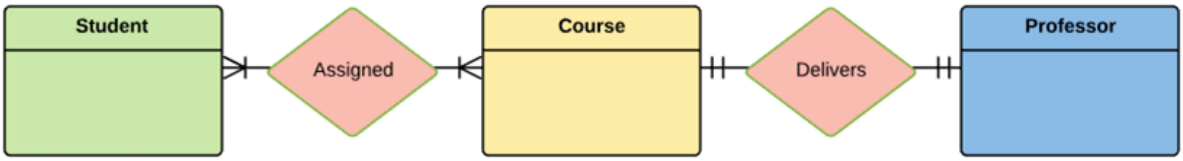
- The student is **assigned** a course
- Professor **delivers** a course



Step 3) Cardinality Identification

For them problem statement we know that,

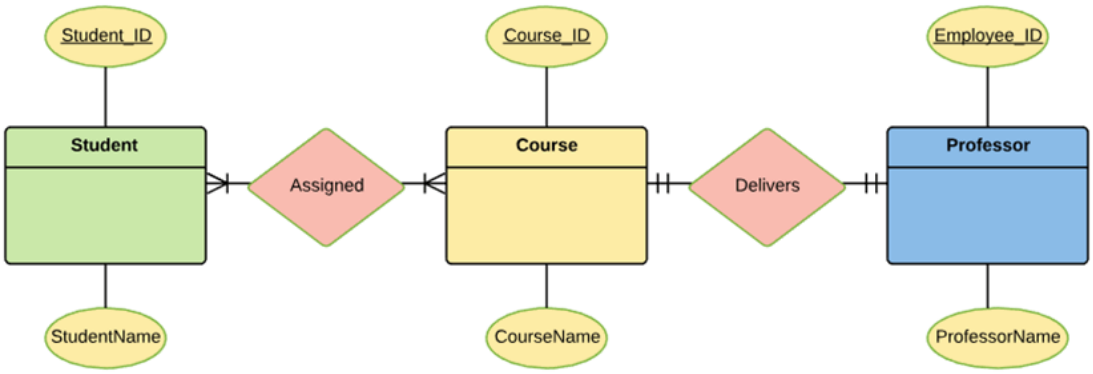
- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



Step 4) Identify Attributes

- You need to study the files, forms, reports, data currently maintained by the organization to identify attributes.
- You can also conduct interviews with various stakeholders to identify entities.
- Initially, it's important to identify the attributes without mapping them to a particular entity.
- Once, you have a list of Attributes, you need to map them to the identified entities.
- Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.
- Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

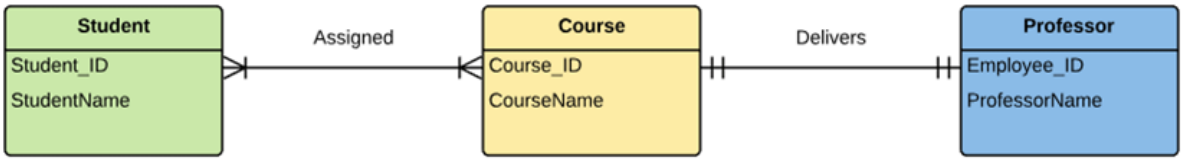
Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

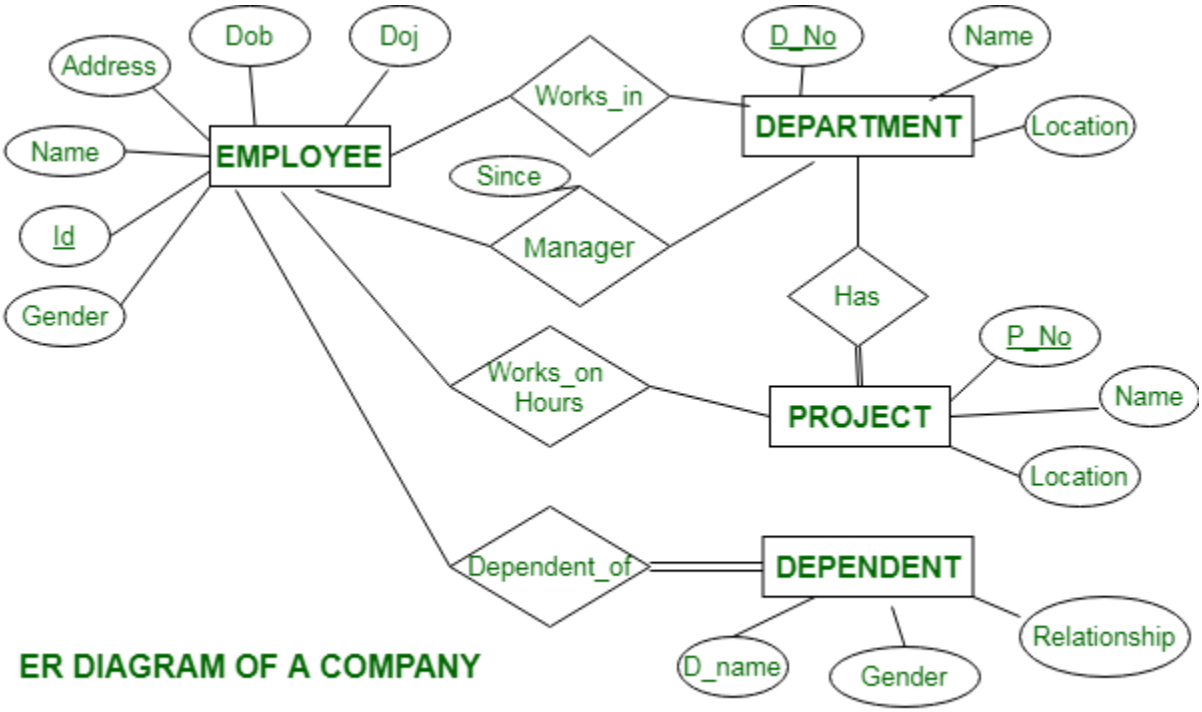


- For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Step 5) Create the ERD Diagram

A more modern representation of Entity Relationship Diagram Example





ER DIAGRAM OF A COMPANY

UNIT – 3 RELATIONAL DATABASE

RELATIONAL MODEL CONCEPT

Relational model can represent as a table with columns and rows. Each row is known as a **tuple.** Each table of the column has a name or **attribute**.

Domain: It contains a set of atomic values that an attribute can take.

Attribute: It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

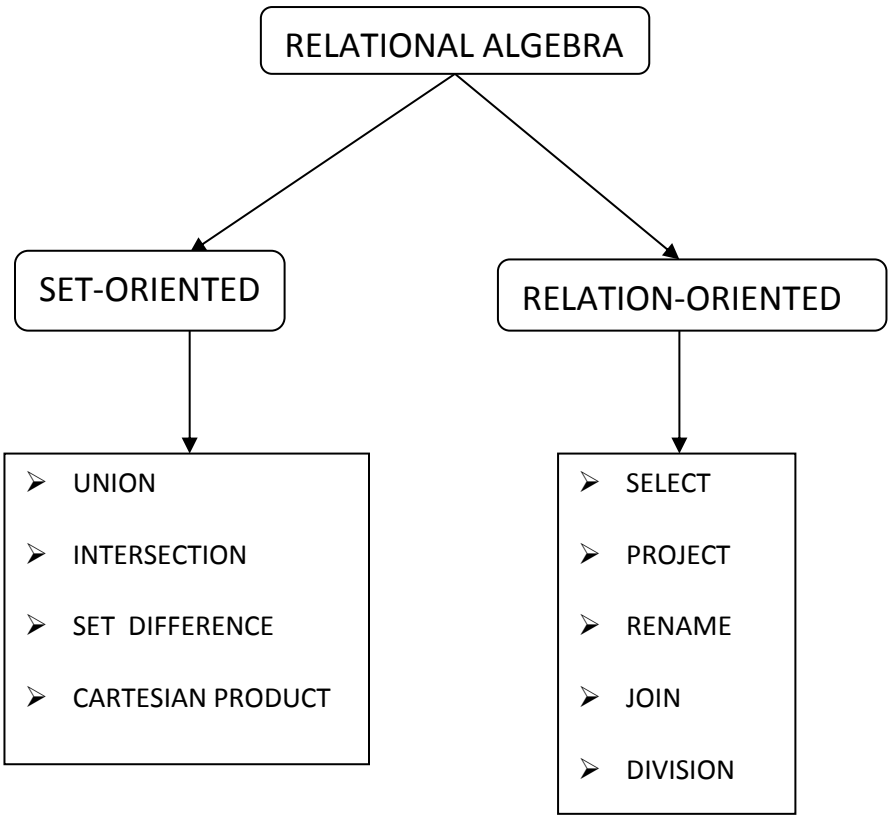
Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- Tuple has no duplicate value
- Order of tuple can have a different sequence

RELATIONAL ALGEBRA

RELATIONAL ALGEBRA is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action. SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations. The relational algebraic operations can be divided into basic set-oriented operations & relational-oriented operations.



SET-ORIENTED OPERATION:-

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

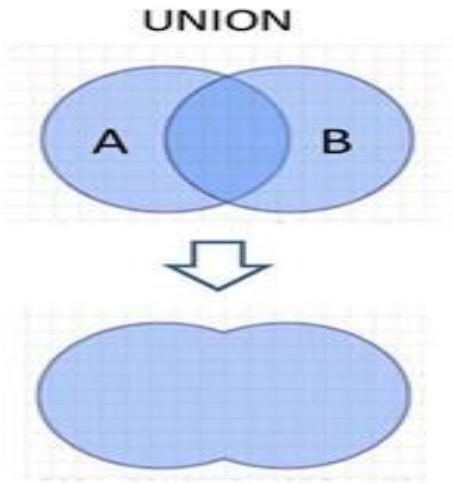
There are 4 different types of SET operations, along with example:

1. UNION
2. INTERSECT

3. MINUS
4. CARTESIAN PRODUCT

1. UNION Operation:

- **UNION** is used to combine the results of two or more SELECT statements.
- However it will eliminate duplicate rows from its result set.
- In case of union, number of columns and data type must be same in both the tables, on which UNION operation is being applied.
- If duplicate values are required in the resultant data, then UNION ALL is used.
- Symbol **'∪'** is used to denote the Union operator.



Example of UNION

A (Name, Age)

Name	Age
Riya	12
Priya	15
Anjali	14

B (Name, Age)

Name	Age
Anu	12
Rena	14

A ∪ B

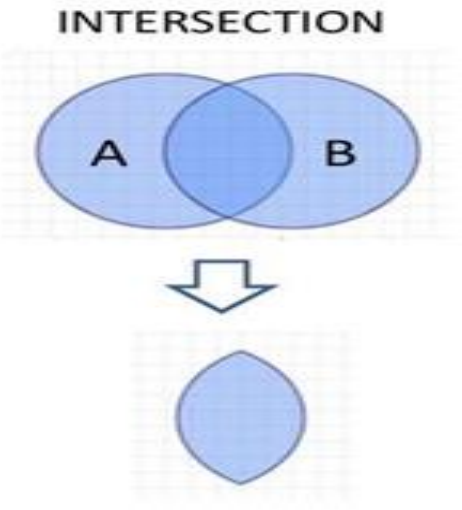
Name	Age
Riya	12
Priya	15
Anjali	14
Anu	12
Rena	14

Union SQL query will be,

```
SELECT * FROM A
UNION
SELECT * FROM B;
```

2. INTERSECT Operation:

- Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.
- In case of Intersect the number of columns and data type must be same.
- Symbol '∩' is used to denote the Intersection operator.
- It has no duplicates and it arranges the data in ascending order by default.



Example of INTERSECT

Name	Age
A	12
B	15
C	18

Name	Age
D	17
E	18
B	15

A ∩ B

Name	Age
B	15

Intersect query will be,

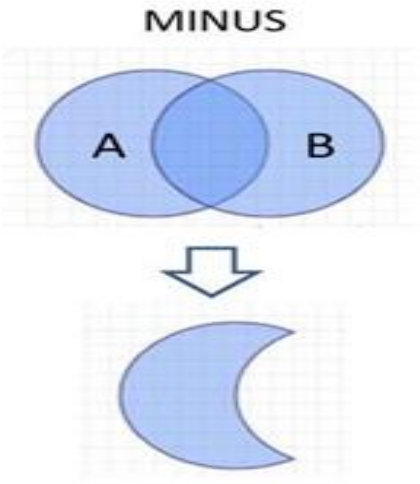
```
SELECT * FROM A
```

INTERSECT

SELECT * FROM **B**;

3. SET DIFFERENCE Operation:

- Set Difference operation is also known as Minus operation.
- The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.
- Symbol ‘ **–** ’ is used to denote the Minus operator.
- It has no duplicates and data arranged in ascending order by default.



Example of MINUS

A (Name, Age)

Name	Age
A	12
B	15
D	14

B (Name, Age)

Name	Age
A	12
C	14
D	14

A - B

Name	Age
B	15

A (Name, Age)

Name	Age
A	12
B	15
D	14

B (Name, Age)

Name	Age
A	12
C	14
D	14

Name	Age
C	14

Minus query will be,

```
SELECT * FROM A
MINUS
SELECT * FROM B;
```

4. CARTESIAN PRODUCT Operation:

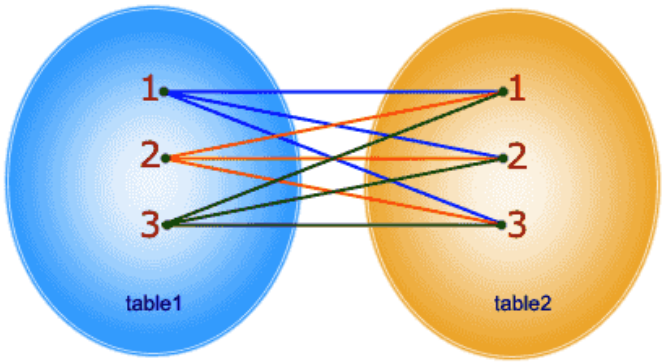
- On applying CARTESIAN PRODUCT on two relations that is on two sets of tuples, it will take every tuple one by one from the left set (relation) and will pair it up with all the tuples in the right set(relation).
- CROSS PRODUCT is a binary set operation means, at a time we can apply the operation on two relations. But the two relations on which we are performing the operations do not have the same type of tuples, which means Union compatibility (or Type compatibility) of the two relations is not necessary.
- **Notation:**

  A ✕ S

  where A and S are the relations,

  the symbol '✕' is used to denote the CROSS PRODUCT operator.
- The result of the Cartesian product of two relations which have x and y columns is a relation that has x + y columns.
- The resulting relation will have one tuple for each combination of tuples one from each participating relation. Hence, if the relations have n and m tuples respectively, then the Cartesian product will have n x m tuples
- We can observe that the number of tuples in STUDENT relation is 3, and the number of tuples in COURSE is 2. So the number of tuples in the resulting relation on performing CROSS PRODUCT is 3*2 = 6.

- We can observe that the number of attributes in STUDENT relation is 4, and the number of attributes in COURSE is 2. So the number of attributes in the resulting relation on performing CROSS PRODUCT is 4+2 = 6.



In CROSS JOIN, each row from 1st table joins with all the rows of another table.
If 1st table contain x rows and y rows in 2nd one the result set will be x * y rows.

Example of CARTESIAN PRODUCT

Student

S_id	Name	Class	Age
1	Andrew	5	25
2	Angel	10	30
3	Anamika	8	35

Course

C_id	C_name
11	Foundation C
21	C++

Student X Course

S_id	Name	Class	Age	C_id	C_name
1	Andrew	5	25	11	Foundation C
1	Andrew	5	25	21	C++
2	Angel	10	30	11	Foundation C
2	Angel	10	30	21	C++
3	Anamika	8	35	11	Foundation C
3	Anamika	8	35	21	C++

RELATIONAL-ORIENTED OPERATION:-

The relational oriented operation is developed specifically for the relational database. Some of the operation in this category is:

1. SELECT
2. PROJECT
3. RENAME
4. JOIN

1. SELECT Operation:

- The **SELECT** operation is a unary operation; it is thus applied to a single relation.
- The selection operation is applied to each tuple individually.
- The SELECT operation is used for selecting a subset of the tuples according to a given selection condition or search criteria.
- The lowercase Greek **Sigma (σ)** Symbol denotes it.
- It is used as an expression to choose tuples which meet the selection condition.
- Select operator selects tuples that satisfy a given predicate.
- Select operation acts like a filter that permits only the tuples that match the specified criteria into the result set.
- The select operation is represented as

  $\sigma_{\text{<selection condition>}}(R)$

  Where

  - **σ** symbol is used to denote the select operator.
  - **<selection condition>** is a Boolean expression specified on the attribute of the relation R.
  - The **Boolean expression** specified in the selection condition is composed of a number of clauses of the form:

    <Attribute name> <comparison operator> <constant value>
  - **<Attribute name>** is the name of an attribute (column) of the relation R.
  - **<comparison operator>** is one of the following operators: =, <, >, ≤, ≥, ≠.

- **<constant value>** is a constant value from the attribute domain.
- Clauses are connected by Boolean operators AND, OR & NOT to form a general selection condition.
- **(R)** is the relational table.

Example of SELECT OPERATION

❖ Select the tuples for all books whose publishing year is 2000.

$\sigma_{year = 2000}$ **(book)**

Select * from book where year = 2000;

❖ Select the tuples for all books whose price is greater than 200.

$\sigma_{price > 200}$ **(book)**

Select * from book where price > 200;

❖ Find all the tuples whose publishing year is 2000 or price is greater than 200.

$\sigma_{(year=2000) \wedge (price > 200)}$ **(book)**

Select * from book where year = 2000 and price > 200 ;

2. PROJECT Operation:

- **Project operation** selects (or chooses) certain attributes discarding other attributes.
- The Project operation is also known as vertical partitioning since it partitions the relation or table vertically discarding other columns or attributes.
- It is a unary operation that returns its argument relation with certain attributes left out.
- Projection is denoted by the Greek letter Pi (π).
- The project operation is represented as :

$$\Pi_A (R)$$

where

- 'A' is the attribute list, it is the desired set of attributes from the attributes of relation(R).
- symbol 'π(pi)' is used to denote the Project operator.
- R is generally a relational algebra expression, which results in a relation.

Example of PROJECT OPERATION

- Display the title and author from the book table.

  $\Pi_{title\ ,\ author}(book)$

  Select title, author from book;

- Display the employee id, name, dept, desg and salary from the emp table.

  $\Pi_{empid\ ,\ name,\ dept,\ desg,\ salary}(Emp)$

  Select empid, name, dept, desg, salary from emp;

3. RENAME Operation:

- The RENAME operation is used to rename the output of a relation.
- Sometimes it is simple and suitable to break a complicated sequence of operations and rename it as a relation with different names.
- Reasons to rename a relation can be many, like –
  - ✓ We may want to save the result of a relational algebra expression as a relation so that we can use it later.
  - ✓ We may want to join a relation with itself, in that case, it becomes too confusing to specify which one of the tables we are talking about, in that case, we rename one of the tables and perform join operations on them.

- 'Rname' operation is denoted with small Greek letter **rho *ρ*.**
- The rename operation is represented as :

  $\rho_{x}(R)$ **or** $\rho_{X(\text{new attribute names})}(R)$

  where .

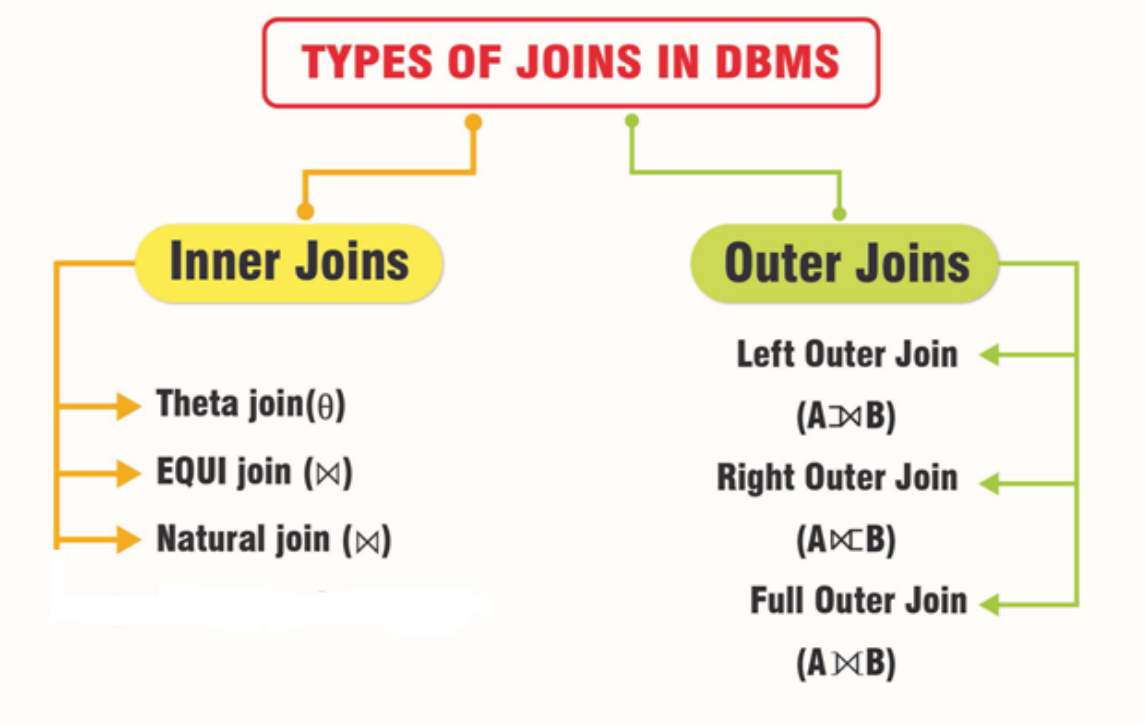
  - the symbol 'ρ' is used to denote the RENAME operator and
  - R is the result of the sequence of operation or expression which is saved with the name X.
  - X is the new relation.

Example of RENAME OPERATION

- ρ_{temp} (book)
- $\rho_{temp\ (bname,\ aname,\ pyear,\ bprice)}$ (book)

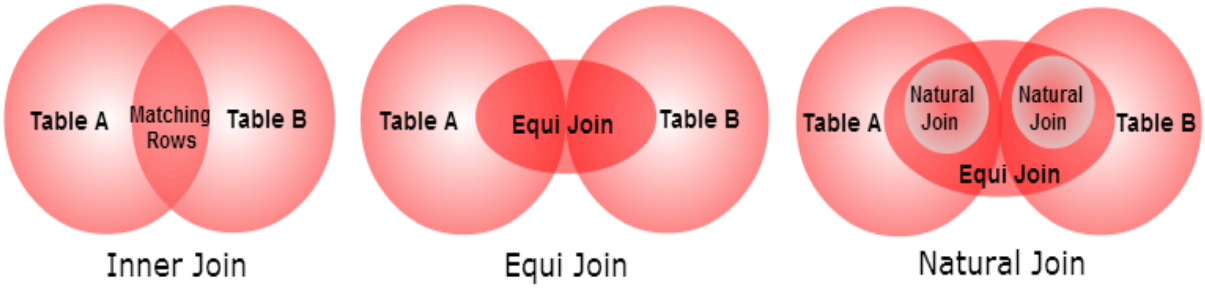
4. JOIN Operation:

- **Join in DBMS** is a binary operation which allows you to combine join product and selection in one single statement.
- The goal of creating a join condition is that it helps you to combine the data from two or more DBMS tables.
- The tables in DBMS are associated using the primary key and foreign keys.
- **Join** is a combination of a Cartesian product followed by a selection process.
- A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.
- The join operati9on is denoted by the symbol ⋈



Inner Joins

- Theta Join, Equijoin, and Natural Join are called inner joins.
- An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation.



Theta (θ) Join

- Theta join combines tuples from different relations provided they satisfy the theta condition.
- The join condition is denoted by the symbol **θ**.
- Notation

  $R1 \bowtie_\theta R2$
- R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is $R1 \cap R2 = \Phi$.
- Theta join can use all kinds of comparison operators.

Student		
SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects	
Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Student_Detail −

$$STUDENT \bowtie_{Student.Std = Subject.Class} SUBJECT$$

Student_detail				
SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

- 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(=).
- When Theta join uses only **equality** comparison operator, it is said to be equijoin.

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT RELATION

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

Natural Join (⋈)

- Natural join does not use any comparison operator.
- It does not concatenate the way a Cartesian product does.
- We can perform a Natural Join only if there is at least one common attribute that exists between two relations.
- In addition, the attributes must have the same name and domain.

- Natural join acts on those matching attributes where the values of attributes in both the relations are same.

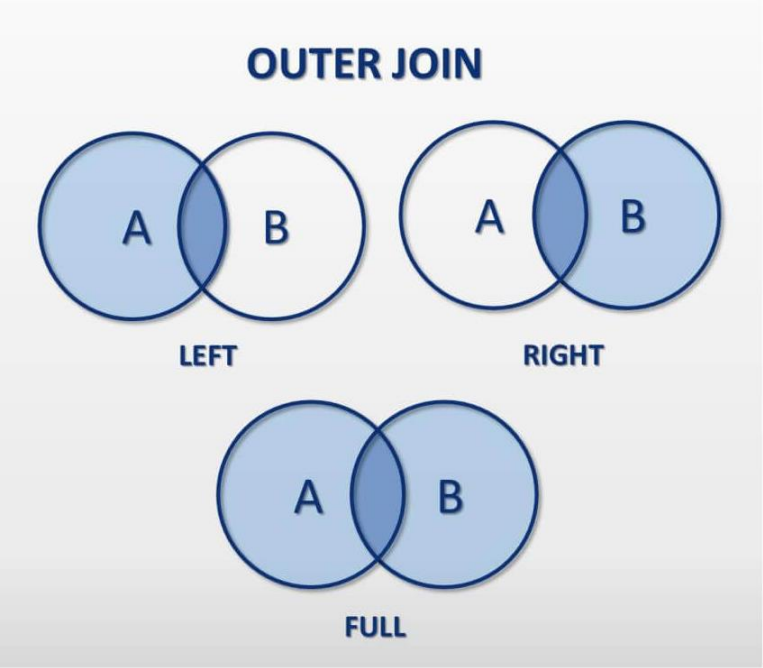
Courses		
CID	**Course**	**Dept**
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD	
Dept	**Head**
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD			
Dept	**CID**	**Course**	**Head**
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

- Theta Join, Equijoin, and Natural Join are called inner joins.
- An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation.
- Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation.
- There are three kinds of outer joins − left outer join, right outer join, and full outer join.



Left Outer Join(R ⟕ S)

- All the tuples from the Left relation, R, are included in the resulting relation.
- If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left	
A	**B**
100	Database

101	Mechanics
102	Electronics

Right	
A	**B**
100	Alex
102	Maya
104	Mira

Courses ⟖ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer Join: (R ⟗ S)

- All the tuples from the Right relation, S, are included in the resulting relation.
- If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R ⟗ S)

- All the tuples from both participating relations are included in the resulting relation.
- If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

UNIT- 4

NORMALIZATION IN RELATIONAL SYSTEM

Functional Dependency

- Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other.
- Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.
- Functional Dependency helps to maintain the quality of data in the database.
- Functional Dependency is represented by $\rightarrow$ (arrow sign).
- A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets.
- It is denoted as **X** $\rightarrow$ **Y**, where X is a set of attributes that is capable of determining the value of Y.
- The attribute set on the left side of the arrow, **X** is called **Determinant**, while on the right side, **Y** is called the **Dependent**.
- There is a functional dependency from x to y or that y is functionally dependent on x.
- The abbreviation for functional dependency is FD or f.d.
- Functional dependencies are used to mathematically express relations among database entities

Example:

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3

From the above table we can conclude some valid functional dependencies:

- **roll_no $\rightarrow$ { name, dept_name, dept_building }**

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

- **roll_no → dept_name**
  Since, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.
- **dept_name → dept_building** ,
  Dept_name can identify the dept_building accurately, since departments with different dept_name will also have a different dept_building
- More valid functional dependencies:
  roll_no → name, {roll_no, name} ⇢ {dept_name, dept_building}, etc.

Here are some invalid functional dependencies:

- **name → dept_name**
  Students with the same name can have different dept_name, hence this is not a valid functional dependency.
- **dept_building → dept_name**
  There can be multiple departments in the same building, For example, in the above table departments ME and EC are in the same building B2, hence dept_building → dept_name is an invalid functional dependency.
- More invalid functional dependencies:
  name → roll_no, {name, dept_name} → roll_no, dept_building → roll_no, etc.

Armstrong's axioms / inference rules of functional dependencies:

1. **Reflexivity:**
   If Y is a subset of X, then X→Y holds by reflexivity rule
   For example, {roll_no, name} → name is valid.

2. **Augmentation:**
   If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.
   For example, If {roll_no, name} → dept_building is valid, hence {roll_no, name, dept_name} → {dept_building, dept_name} is also valid.

3. **Transitivity**:

If X → Y and Y → Z are both valid dependencies, then X→Z is also valid by the Transitivity rule.

For example, roll_no → dept_name & dept_name → dept_building, then roll_no → dept_building is also valid.

Types of Functional dependencies in DBMS:

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

1. Trivial Functional Dependency

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant. i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

- Here, **{roll_no, name} → name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}**
- Similarly, **roll_no → roll_no** is also an example of trivial functional dependency.

2. Non-trivial Functional Dependency

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

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

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

3. Multivalued Functional Dependency

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other.**i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

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

4. Transitive Functional Dependency

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

For example,

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

- Here, **enrol_no → dept** and **dept → building_no**,

Hence, according to the axiom of transitivity, **enrol_no** → **building_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

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 design

Equivalence of Sets of Functional Dependencies

- A set of functional dependencies E is covered by a set of functional dependencies F—or alternatively, F is said to cover E—if every FD in E is also in ; that is, if every dependency in E can be inferred from F.
- Two sets of functional dependencies E and F are equivalent if $E^+ = F^+$. Hence, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions E covers F and F covers E hold.

Minimal Sets of Functional Dependencies

- A set of functional dependencies F is minimal if it satisfies the following conditions:

  1. Every dependency in F has a single attribute for its right-hand side.

  2. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.

  3. We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

- A minimal set of dependencies as being a set of dependencies in a standard or canonical form and with no redundancies.
- **Condition 1** ensures that every dependency is in a canonical form with a single attribute on the right-hand side.

- **Conditions 2 and 3** ensure that there are no redundancies in the dependencies either by having redundant attributes on the left-hand side of a dependency (Condition 2), or by having a dependency that can be inferred from the remaining FDs in F (Condition 3).
- A minimal cover of a set of functional dependencies F is a minimal set of dependencies that is equivalent to F.
- Unfortunately, there can be several minimal covers for a set of functional dependencies

Keys in DBMS

- **KEYS in DBMS** is an attribute or set of attributes which helps you to identify a row (tuple) in a relation(table).
- They allow you to find the relation between two tables.
- Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.
- Key is also helpful for finding unique record or row from the table.
- Database key is also helpful for finding unique record or row from the table.

Why we need a Key?

Here are some reasons for using sql key in the DBMS system.

- Keys help you to identify any row of data in a table.
- In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges.
- Allows you to establish a relationship between and identify the relation between tables.
- Help you to enforce identity and integrity in the relationship.

Types of keys

1. Super Key-

- A super key is a set of attributes that can identify each tuple uniquely in the given relation.
- A super key is not restricted to have any specific number of attributes.
- Thus, a super key may consist of any number of attributes.

- All the attributes in a super key are definitely sufficient to identify each tuple uniquely in the given relation but all of them may not be necessary.

Example-

Consider the following Student schema-

Student (roll , name , sex , age , address , class , section)

Given below are the examples of super keys since each set can uniquely identify each student in the Student table-

- (roll , name , sex , age , address , class , section)
- (class , section , roll)
- (class , section , roll , sex)
- (name , address)

2. Candidate Key-

- A minimal super key is called as a candidate key.
- A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called as a candidate key.
- All the attributes in a candidate key are sufficient as well as necessary to identify each tuple uniquely.
- Removing any attribute from the candidate key fails in identifying each tuple uniquely.
- The value of candidate key must always be unique.
- The value of candidate key can never be NULL.
- It is possible to have multiple candidate keys in a relation.
- Those attributes which appears in some candidate key are called as **prime attributes**.

Example-

Consider the following Student schema-

Student (roll , name , sex , age , address , class , section)

Given below are the examples of candidate keys since each set consists of minimal attributes required to identify each student uniquely in the Student table-

- (class , section , roll)
- (name , address)

3. Primary Key-

- A primary key is a candidate key that the database designer selects while designing the database.

- The value of primary key can never be NULL.
- The value of primary key must always be unique.
- The values of primary key can never be changed i.e. no updation is possible.
- The value of primary key must be assigned when inserting a record.
- A relation is allowed to have only one primary key.

4. Alternate Key-

- Candidate keys that are left unimplemented or unused after implementing the primary key are called as alternate keys.

5. Foreign Key-

- An attribute ‘X’ is called as a foreign key to some other attribute ‘Y’ when its values are dependent on the values of attribute ‘Y’.

- The attribute ‘X’ can assume only those values which are assumed by the attribute ‘Y’.
- Here, the relation in which attribute ‘Y’ is present is called as the **referenced relation**.
- The relation in which attribute ‘X’ is present is called as the **referencing relation**.
- The attribute ‘Y’ might be present in the same table or in some other table.

NOTES-

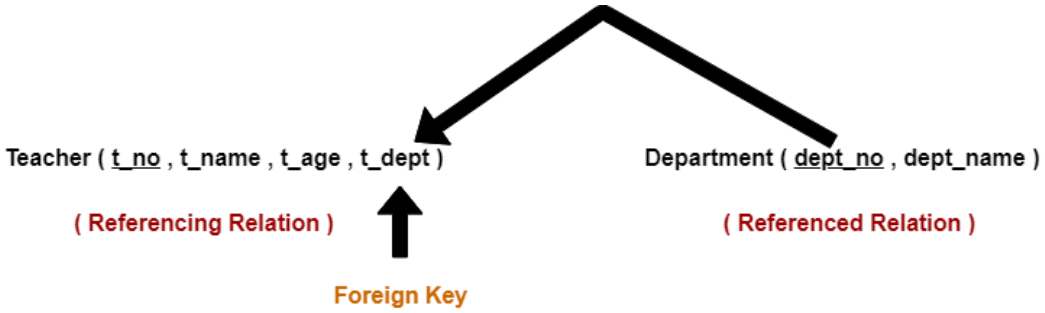
- Foreign key references the primary key of the table.

- Foreign key can take only those values which are present in the primary key of the referenced relation.
- Foreign key may have a name other than that of a primary key.
- Foreign key can take the NULL value.

- There is no restriction on a foreign key to be unique.
- In fact, foreign key is not unique most of the time.
- Referenced relation may also be called as the master table or primary table.
- Referencing relation may also be called as the foreign table.

Example-

Consider the following two schemas-



Here, t_dept can take only those values which are present in dept_no in Department table since only those departments actually exist.

6. Partial Key-

- Partial key is a key using which all the records of the table can not be identified uniquely.
- However, a bunch of related tuples can be selected from the table using the partial key.

Example-

Consider the following schema-

Department (Emp_no , Dependent_name , Relation)

Emp_no	Dependent_name	Relation
E1	Suman	Mother
E1	Ajay	Father
E2	Vijay	Father
E2	Ankush	Son

Here, using partial key Emp_no, we can not identify a tuple uniquely but we can select a bunch of tuples from the table.

7. Composite Key-

- A primary key comprising of multiple attributes and not just a single attribute is called as a composite key.

8. Unique Key-

Unique key is a key with the following properties-

- It is unique for all the records of the table.
- Once assigned, its value can not be changed i.e. it is non-updatable.
- It may have a NULL value.

Example-

The best example of unique key is **Adhaar Card Numbers**.

- The Adhaar Card Number is unique for all the citizens (tuples) of India (table).
- If it gets lost and another duplicate copy is issued, then the duplicate copy always has the same number as before.
- Thus, it is non-updatable.
- Few citizens may not have got their Adhaar cards, so for them its value is NULL.

9. Surrogate Key-

Surrogate key is a key with the following properties-

- It is unique for all the records of the table.
- It is updatable.
- It can not be NULL i.e. it must have some value.

Example-

Mobile Number of students in a class where every student owns a mobile phone.

10. Secondary Key-

- Secondary key is required for the indexing purpose for better and faster searching.

Data Anomalies

- Anomalies are caused when there is too much redundancy in the database's information.
- Anomalies can often be caused when the tables that make up the database suffer from poor construction.
- So, what does "poor construction" mean?

- Poor table design will become evident if, when the designer creates the database, he doesn't identify the entities that depend on each other for existence, like the rooms of a hotel and the hotel, and then minimize the chance that one would ever exist independent of the other.
- The normalization process was created largely in order to reduce the negative effects of creating tables that will introduce anomalies into the database.
- Generally this is removed by the process of normalization which is performed by splitting / joining of tables.
- There are three types of **Data Anomalies**:
    - ✓ Update Anomalies
    - ✓ Insertion Anomalies
    - ✓ Deletion Anomalies.

Example:

Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, e_name for storing employee's name, e_address for storing employee's address, and e_dept for storing the department details in which the employee works. At some point in time the table looks like this: e_id e_name e_address e_dept

e_id	e_name	e_address	e_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890

166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Update anomaly:

- When duplicated data is updated at one instance and not across all instances where it was duplicated. That's an update anomaly .
- In the above table, we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick 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, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly:

- An Insert Anomaly occurs when attributes cannot be inserted into the database without the presence of other attributes.
- 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 the e_dept field doesn't allow nulls.

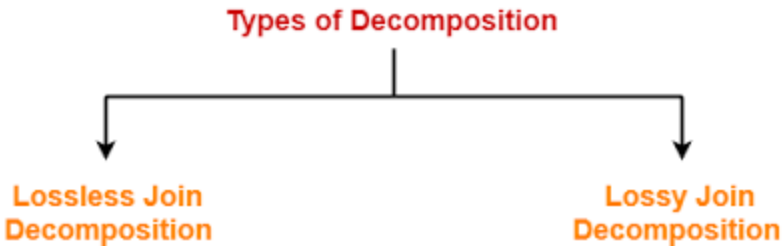
Delete anomaly:

- Deletion of some attribute which causes deletion of other attributes is deletion anomaly.
- Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having e_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

Decomposition

Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables.

The following are the types –



Lossless Decomposition

Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not lose from the relation when decomposed. The join would result in the same original relation.

Let us see an example −

<EmpInfo>

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

<EmpDetails>

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	Emp_ID	Dept_Name
Dpt1	E001	Operations
Dpt2	E002	HR
Dpt3	E003	Finance

Now, Natural Join is applied on the above two tables −

The result will be −

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Therefore, the above relation had lossless decomposition i.e. no loss of information.

Lossy Decomposition

As the name suggests, when a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved.

Let us see an example −

<EmpInfo>

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables −

<EmpDetails>

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	Dept_Name
Dpt1	Operations
Dpt2	HR
Dpt3	Finance

Now, you won't be able to join the above tables, since **Emp_ID** isn't part of the **DeptDetails** relation.Therefore, the above relation has lossy decomposition.

Normalization

- The normalization process, as first proposed by Codd in 1972.
- The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations.
- Initially, Codd proposed three normal forms, which he called first, second, and third normal form.
- A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd.
- All these normal forms are based on the functional dependencies among the attributes of a relation.
- Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies.
- **Normalization** is the process of minimizing **redundancy** from a relation or set of relations.
- Redundancy in relation may cause insertion, deletion and updation anomalies.
  So, it helps to minimize the redundancy in relations.
- **Normal forms** are used to eliminate or reduce redundancy in database tables.
- Normalization rules divides larger tables into smaller tables and links them using relationships.
- The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

Example:-

EMPLOYEE (UNNORMALIZED)				
EMP_NO	**NAME**	**DEPT_NO**	**DEPT_NAME**	**SKILLS**
1	KEVIN	201	R&D	C,PERL,JAVA
2	JOHN	224	IT	LINUX,C++
3	JACK	201	R&D	DB2,ORACLE,JAVA

First Normal Form (1NF)

The table to be in the First Normal Form, it should follow the following rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names, and the order in which data is stored, does not matter.

EMPLOYEE (1NF)				
EMP_NO	**NAME**	**DEPT_NO**	**DEPT_NAME**	**SKILLS**
1	KEVIN	201	R&D	C
1	KEVIN	201	R&D	PERL
1	KEVIN	201	R&D	JAVA
2	JOHN	224	IT	LINUX
2	JOHN	224	IT	C++
3	JACK	201	R&D	DB2
3	JACK	201	R&D	ORACLE
3	JACK	201	R&D	JAVA

Second Normal Form (2NF)

The table to be in the Second Normal Form, then

1. It should be in the First Normal form.
2. It should not have Partial Dependency. i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
3. **Partial Dependency –** If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.
4. Every attribute must be functionally dependent on the primary key
5. Any non-dependent attributes are moved into a smaller subset table
6. It improves data integrity. It prevents update ,insert & delete anomalies.

EMPLOYEE (1NF)				
EMP_NO	**NAME**	**DEPT_NO**	**DEPT_NAME**	**SKILLS**
1	KEVIN	201	R&D	C
1	KEVIN	201	R&D	PERL
1	KEVIN	201	R&D	JAVA
2	JOHN	224	IT	LINUX
2	JOHN	224	IT	C++
3	JACK	201	R&D	DB2
3	JACK	201	R&D	ORACLE
3	JACK	201	R&D	JAVA

- Here in the above table **name, dept_no & dept_name** are functionally dependent on emp_no.

  emp_no → (name , dept_no, dept_name)

- **Skill** is not functionally dependent on **emp_no** since it is not unique to each **emp_no**
- So the second normal tables are

EMPLOYEE (2 NF)			
EMP_NO	**NAME**	**DEPT_NO**	**DEPT_NAME**
1	KEVIN	201	R&D
2	JOHN	224	IT
3	JACK	201	R&D

SKILLS (2NF)	
EMP_NO	**SKILLS**
1	C
1	PERL
1	JAVA
2	LINUX
2	C++

3	DB2
3	ORACLE
3	JAVA

Third Normal Form (3NF)

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

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.
3. **Transitive dependency –** If A->B and B->C are two FDs then A->C is called transitive dependency. That means two separate entities exist within one table.
4. **Any transitive dependencies are moved into a smaller subset table.**
5. It improves data integrity. It also prevents update ,insert & delete anomalies.

EMPLOYEE (2 NF)			
EMP_NO	**NAME**	**DEPT_NO**	**DEPT_NAME**
1	KEVIN	201	R&D
2	JOHN	224	IT
3	JACK	201	R&D

- Here in the above table **dept_no & dept_name** are functionally dependent on **emp_no.** however dept can be considered as a separate entity.

EMPLOYEE (3 NF)		
EMP_NO	**NAME**	**DEPT_NO**
1	KEVIN	201
2	JOHN	224
3	JACK	201

DEPARTMENT(3NF)	
DEPT_NO	**DEPT_NAME**
201	R&D
224	IT

Boyce and Codd Normal Form (BCNF)

- **Boyce and Codd Normal Form** is a higher version of the Third Normal form.
- This form deals with certain type of anomaly that is not handled by 3NF.
- A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.
- For a table to be in BCNF, following conditions must be satisfied:

    1. R must be in 3rd Normal Form
    2. and, for each functional dependency (X → Y), X should be a super Key.

- The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.

Example

Below we have a college enrolment table with columns student_id, subject and professor.

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

In the table above:

- One student can enrol for multiple subjects. For example, student with **student_id** 101, has opted for subjects - Java & C++
- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

- student_id, subject together form the primary key, because using student_id and subject, we can find all the columns of the table.
- One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.
- Hence, there is a dependency between subject and professor here, where subject depends on the professor name.
- This table satisfies the **1st 2nd & 3rd Normal form**
- But this table is not in **Boyce-Codd Normal Form**.

Why this table is not in BCNF?

- In the table above, student_id, subject form primary key, which means subject column is a **prime attribute**.
- But, there is one more dependency, professor → subject.
- And while subject is a prime attribute, professor is a **non-prime attribute**, which is not allowed by BCNF.

How to satisfy BCNF?

- To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **student** table and **professor** table.
- Below we have the structure for both the tables.

Student Table

student_id	p_id
101	1
101	2
and so on...	

And, **Professor Table**

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++
and so on...		

And now, this relation satisfy Boyce-Codd Normal Form. In the next tutorial we will learn about the **Fourth Normal Form**.

Objectives of database normalization

- To correct duplicate data and database anomalies.
- To avoid creating and updating any unwanted data connections and dependencies.
- To prevent unwanted deletions of data.
- To optimize storage space.
- To reduce the delay and complexity of checking databases when new types of data need to be introduced.
- To facilitate the access and interpretation of data to users and applications that make use of the databases.

Advantages of normalization

- **Updates run quickly** due to no data being duplicated in multiple locations.
- **Inserts run quickly** since there is only a single insertion point for a piece of data and no duplication is required.
- **Tables are typically smaller** than the tables found in non-normalized databases. This usually allows the tables to fit into the buffer, thus offering faster performance.
- **Data integrity and consistency is an absolute must** if the database must be ACID compliant. A normalized database helps immensely with such an undertaking.

Disadvantages of normalization

- **Since data is not duplicated, table joins are required**. This makes queries more complicated, and thus read times are slower.

- **Since joins are required, indexing does not work as efficiently**. Again, this makes read times slower because the joins don't typically work well with indexing.

Summary of Normalization

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

UNIT – 5

STRUCTURE QUERY LANGUAGE

SQL

- SQL stands for **Structured Query Language**.
- It is designed for managing data in a relational database management system (RDBMS).
- It is pronounced as S-Q-L or sometime **See-Qwell**.
- SQL is a database language, it is used for database creation, deletion, fetching rows, and modifying rows, etc.
- SQL is based on relational algebra and tuple relational calculus.

All DBMS like MySQL, Oracle, MS Access, Sybase, Informix, PostgreSQL, and SQL Server use SQL as standard database language.

Why SQL is required

SQL is required:

- To create new databases, tables and views
- To insert records in a database
- To update records in a database
- To delete records from a database
- To retrieve data from a database

What SQL does

- With SQL, we can query our database in several ways, using English-like statements.
- With SQL, a user can access data from a relational database management system.

- It allows the user to describe the data.
- It allows the user to define the data in the database and manipulate it when needed.
- It allows the user to create and drop database and table.
- It allows the user to create a view, stored procedure, function in a database.
- It allows the user to set permission on tables, procedures, and views.

Advantages of SQL:

SQL has many advantages which makes it popular and highly demanded. It is a reliable and efficient language used for communicating with the database. Some advantages of SQL are as follows:

1. **Faster Query Processing –**
   Large amount of data is retrieved quickly and efficiently. Operations like Insertion, deletion, manipulation of data is also done in almost no time.
2. **No Coding Skills –**
   For data retrieval, large number of lines of code is not required. All basic keywords such as SELECT, INSERT INTO, UPDATE, etc are used and also the syntactical rules are not complex in SQL, which makes it a user-friendly language.
3. **Standardised Language –**
   Due to documentation and long establishment over years, it provides a uniform platform worldwide to all its users.
4. **Portable –**
   It can be used in programs in PCs, server, laptops independent of any platform (Operating System, etc). Also, it can be embedded with other applications as per need/requirement/use.
5. **Interactive Language –**
   Easy to learn and understand, answers to complex queries can be received in seconds.

Disadvantages of SQL:

Although SQL has many advantages, still there are a few disadvantages. Various Disadvantages of SQL are as follows:

1. **Complex Interface –**

SQL has a difficult interface that makes few users uncomfortable while dealing with the database.

2. **Cost –**
   Some versions are costly and hence, programmers cannot access it.
3. **Partial Control –**
   Due to hidden business rules, complete control is not given to the database.

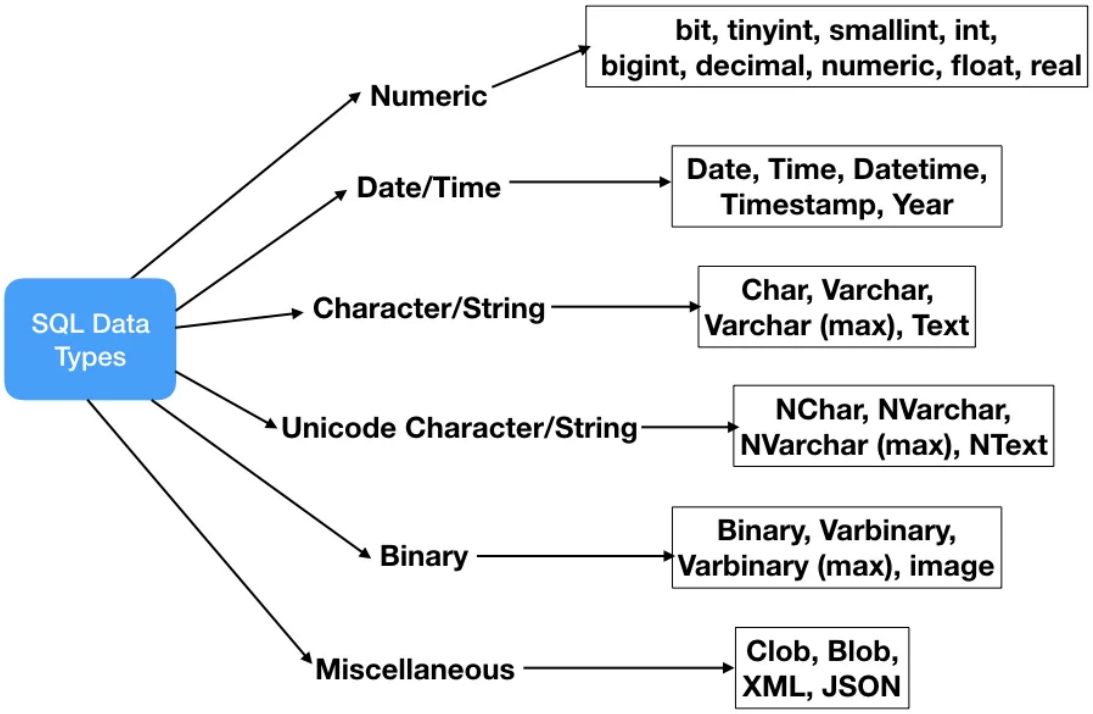
Applications of SQL :

- SQL is used by developers and DBAs (Database Administrators) in writing Data Integration Scripts.
- It is used to deal with analytical queries to analyze the data and get instincts from it.
- Retrieving Information
- Modification/Manipulation of data and database table such as Insertion, Deletion and Updation.

SQL Data Types

SQL data types can be broadly divided into following categories.

1. Numeric data types such as **int, tinyint, bigint, float, real, etc.**
2. Date and Time data types such as **Date, Time, Datetime, etc.**
3. Character and String data types such as **char, varchar, text, etc.**
4. Unicode character string data types, for example **nchar, nvarchar, ntext, etc.**
5. Binary data types such as **binary, varbinary, etc.**
6. Miscellaneous data types – **clob, blob, xml, cursor, table, etc.**



SQL Numeric Data Types

Datatype	From	To
bit	0	1
tinyint	0	255
smallint	-32,768	32,767
int	-2,147,483,648	2,147,483,647
bigint	-9,223,372,036, 854,775,808	9,223,372,036, 854,775,807
decimal	-10^38 +1	10^38 -1

numeric	-10^38 +1	10^38 -1
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

SQL Date and Time Data Types

Datatype	Description
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS
TIMESTAMP	Stores number of seconds passed since the Unix epoch ('1970-01-01 00:00:00' UTC)
YEAR	Stores year in 2 digits or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069.

SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with a maximum length of 8,000 characters
VARCHAR	Variable-length storage with a maximum length of 8,000 characters

VARCHAR(max)	Variable-length storage with provided max characters, not supported in MySQL
TEXT	Variable-length storage with maximum size of 2GB data

Note that all the above data types are for character stream, they should not be used with Unicode data.

SQL Unicode Character and String Data Types

Datatype	Description
NCHAR	Fixed length with maximum length of 4,000 characters
NVARCHAR	Variable-length storage with a maximum length of 4,000 characters
NVARCHAR(max)	Variable-length storage with provided max characters
NTEXT	Variable-length storage with a maximum size of 1GB data

Note that above data types are not supported in MySQL database.

SQL Binary Data Types

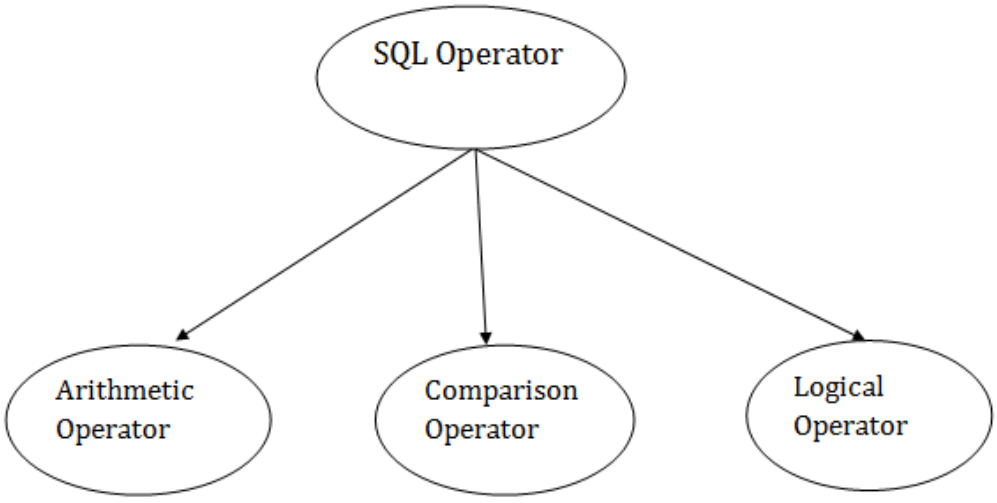
Datatype	Description
BINARY	Fixed length with a maximum length of 8,000 bytes
VARBINARY	Variable-length storage with a maximum length of 8,000 bytes

VARBINARY(max)	Variable-length storage with provided max bytes
IMAGE	Variable-length storage with maximum size of 2GB binary data

SQL Miscellaneous Data Types

Datatype	Description
CLOB	Character large objects that can hold up to 2GB
BLOB	For binary large objects
XML	for storing XML data
JSON	for storing JSON data

SQL OPERATOR



SQL Arithmetic Operators

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
+	It adds the value of both operands.	a+b will give 30
-	It is used to subtract the right-hand operand from the left-hand operand.	a-b will give 10
*	It is used to multiply the value of both operands.	a*b will give 200
/	It is used to divide the left-hand operand by the right-hand operand.	a/b will give 2
%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.	a%b will give 0

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
=	It checks if two operands values are equal or not, if the values are queal then condition becomes true.	(a=b) is not true
!=	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.	(a!=b) is true
<>	It checks if two operands values are equal or not, if values are not equal then condition becomes true.	(a<>b) is true

>	It checks if the left operand value is greater than right operand value, if yes then condition becomes true.	(a>b) is not true
<	It checks if the left operand value is less than right operand value, if yes then condition becomes true.	(a<b) is true
>=	It checks if the left operand value is greater than or equal to the right operand value, if yes then condition becomes true.	(a>=b) is not true
<=	It checks if the left operand value is less than or equal to the right operand value, if yes then condition becomes true.	(a<=b) is true
!<	It checks if the left operand value is not less than the right operand value, if yes then condition becomes true.	(a!=b) is not true
!>	It checks if the left operand value is not greater than the right operand value, if yes then condition becomes true.	(a!>b) is true

SQL Logical Operators

There is the list of logical operator used in SQL:

Operator	Description
ALL	It compares a value to all values in another value set.
AND	It allows the existence of multiple conditions in an SQL statement.
ANY	It compares the values in the list according to the condition.

BETWEEN	It is used to search for values that are within a set of values.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statements.
EXISTS	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

ORACLE FUNCTIONS:

❖ Built in Functions

There are two types of functions in Oracle.

1) Single Row Functions:

Single row or Scalar functions return a value for every row that is processed in a query.

2) Group Functions:

These functions group the rows of data based on the values returned by the query. This is discussed in SQL GROUP Functions. The group functions are used to calculate aggregate values like total or average, which return just one total or one average value after processing a group of rows.

There are four types of **single row functions**. They are:

- **Numeric Functions:**
- **Character or Text Functions:**
- **Date Functions:**
- **Conversion Functions:**

You can combine more than one function together in an expression. This is known as nesting of functions.

1) Numeric Functions:

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

FUNCTION NAME	PURPOSE	EXAMPLES	RETURN VALUE
ABS (x)	Absolute value of the number '*x*'	ABS (1) ABS (-1)	1 -1
CEIL (x)	Integer value that is Greater than or equal to the number '*x*'	CEIL (2.83) CEIL (2.49) CEIL (-1.6)	3 3 -1
FLOOR (x)	Integer value that is Less than or equal to the number '*x*'	FLOOR (2.83) FLOOR (2.49) FLOOR (-1.6)	2 2 -2
TRUNC (x, y)	Truncates value of number '*x*' up to '*y*' decimal places	ROUND (125.456, 1) ROUND (125.456, 0) ROUND (124.456, -1)	125.4 125 120
ROUND (x, y)	Rounded off value of the number '*x*' up to the number '*y*' decimal places	TRUNC (140.234, 2) TRUNC (-54, 1) TRUNC (5.7) TRUNC (142, -1)	140.23 54 5 140

2) Character or Text Functions:

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

Function Name	Return Value	Examples	Return Value
LOWER (string_value)	All the letters in *'string_value'* is converted to lowercase.	LOWER('Good Morning')	good morning
UPPER (string_value)	All the letters in *'string_value'* is converted to uppercase.	UPPER('Good Morning')	GOOD MORNING

INITCAP (string_value)	All the letters in *'string_value'* is converted to mixed case.	INITCAP('GOOD MORNING')	Good Morning
LTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the left of *'string_value'*.	LTRIM ('Good Morning', 'Good)	Morning
RTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the right of *'string_value'* .	RTRIM ('Good Morning', ' Morning')	Good
TRIM (trim_text FROM string_value)	All occurrences of *'trim_text'* from the left and right of *'string_value'* , *'trim_text'* can also be only one character long .	TRIM ('o' FROM 'Good Morning')	Gd Mrning
SUBSTR (string_value, m, n)	Returns *'n'* number of characters from *'string_value'* starting from the '*m'* position.	SUBSTR ('Good Morning', 6, 7)	Morning
LENGTH (string_value)	Number of characters in *'string_value'* in returned.	LENGTH ('Good Morning')	12
LPAD (string_value, n, pad_value)	Returns '*string_value'* left-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.	LPAD ('Good', 6, '*')	**Good
RPAD (string_value, n, pad_value)	Returns '*string_value'* right-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.	RPAD ('Good', 6, '*')	Good**

3) Date Functions:

These are functions that take values that are of datatype DATE as input and return values of datatypes DATE, except for the MONTHS_BETWEEN function, which returns a number as output. Few date functions are as given below.

Function Name	Return Value

ADD_MONTHS (date, n)	Returns a date value after adding *'n'* months to the date *'x'*.
MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date *'x'* rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the *'date_format'*.
TRUNC (x, date_format)	Returns the date *'x'* lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.
NEXT_DAY (x, week_day)	Returns the next date of the *'week_day'* on or after the date *'x'* occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date *'x'* specified.
SYSDATE	Returns the systems current date and time.
NEW_TIME (x, zone1, zone2)	Returns the date and time in zone2 if date 'x' represents the time in zone1.

The below table provides the examples for the above functions

Function Name	Examples	Return Value
ADD_MONTHS ( )	ADD_MONTHS ('16-Sep-81', 3)	16-Dec-81
MONTHS_BETWEEN( )	MONTHS_BETWEEN ('16-Sep-81', '16-Dec-81')	3
NEXT_DAY( )	NEXT_DAY ('01-Jun-08', 'Wednesday')	04-JUN-08
LAST_DAY( )	LAST_DAY ('01-Jun-08')	30-Jun-08
NEW_TIME( )	NEW_TIME ('01-Jun-08', 'IST', 'EST')	31-May-08

4) Conversion Functions:

These are functions that help us to convert a value in one form to another form. For Ex: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE. Few of the conversion functions available in oracle are:

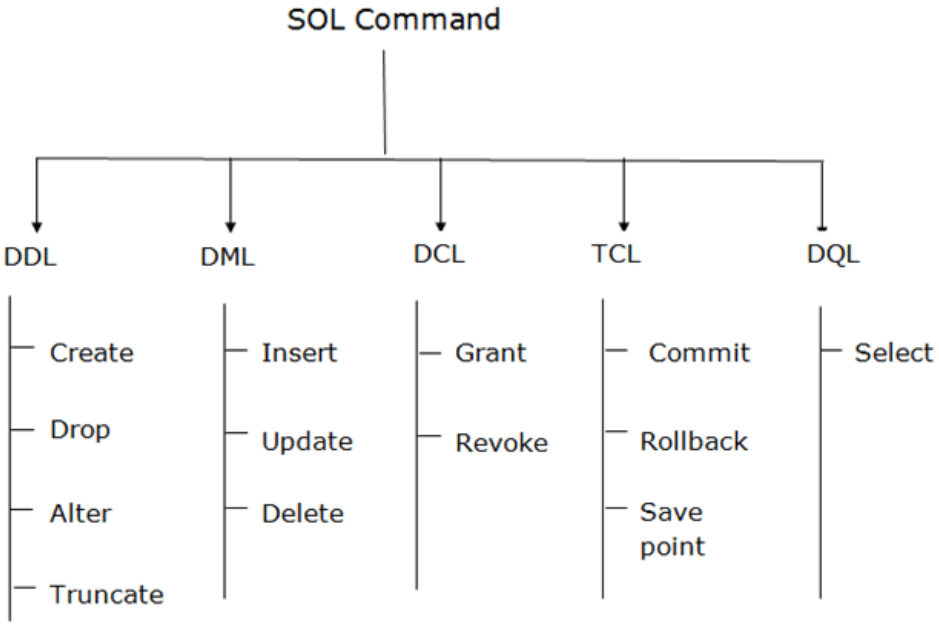
Function Name	Return Value	Examples	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.	TO_CHAR (3000, '$9999') TO_CHAR (SYSDATE, 'Day, Month YYYY')	$3000 Monday, June 2008
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by *'date_format'*.	TO_DATE ('01-Jun-08')	01-Jun-08
NVL (x, y)	If *'x'* is NULL, replace it with *'y'*. *'x'* and *'y'* must be of the same datatype.	NVL (null, 1)	1
DECODE (a, b, c, d, e, default_value)	Checks the value of *'a'*, if *a = b*, then returns *'c'*. If *a = d*, then returns *'e'*. Else, returns *default_value*.		

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.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



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[,....]);

Example:

- CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

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

Syntax

- DROP TABLE ;

Example

- DROP TABLE EMPLOYEE;

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 probably 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....);

EXAMPLE

- ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
- ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

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;

Example:

- TRUNCATE TABLE EMPLOYEE;

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);

For example:

- INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

b. 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 CON DITION]

For example:

- UPDATE students
  SET User_Name = 'Sonoo'
  WHERE Student_Id = '3'

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

Syntax:

- DELETE FROM table_name [WHERE condition];

For example:

- DELETE FROM javatpoint
  WHERE Author="Sonoo";

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:

- Grant
- Revoke

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

Example

- GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

- REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

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;

Example:

- DELETE FROM CUSTOMERS
  WHERE AGE = 25;
- COMMIT;

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

Syntax:

- ROLLBACK;

Example:

- DELETE FROM CUSTOMERS

WHERE AGE = 25;

- 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;

For example:

- SELECT emp_name
  FROM employee
  WHERE age > 20;

SQL SELECT DISTINCT

The **SQL DISTINCT command** is used with SELECT key word to retrieve only distinct or unique data.

In a table, there may be a chance to exist a duplicate value and sometimes we want to retrieve only unique values. In such scenarios, SQL SELECT DISTINCT statement is used.

Syntax:

- **SELECT DISTINCT** column_name ,column_name
  FROM table_name;

For example:

Student_Name	Gender	Mobile_Number	HOME_TOWN
Rahul Ojha	Male	7503896532	Lucknow
Disha Rai	Female	9270568893	Varanasi
Sonoo Jaiswal	Male	9990449935	Lucknow

Here is a table of students from where we want to retrieve distinct information For example: distinct home-town.

- **SELECT DISTINCT** home_town
  FROM students

HOME_TOWN
Lucknow
Varanasi

SQL ORDER BY Clause

- The SQL ORDER BY clause is used for sorting data in ascending and descending order based on one or more columns.
- Some databases sort query results in ascending order by default.

Syntax:

❖ **SELECT** expressions
FROM tables
WHERE conditions
ORDER BY expression [**ASC** | **DESC**];

For example:

Let us take a CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Himani gupta	21	Modinagar	22000
2	Shiva tiwari	22	Bhopal	21000
3	Ajeet bhargav	45	Meerut	65000
4	Ritesh yadav	36	Azamgarh	26000
5	Balwant singh	45	Varanasi	36000
6	Mahesh sharma	26	Mathura	22000

This is an example that would sort the result in ascending order by NAME and SALARY.

❖ **SELECT** * **FROM** CUSTOMERS
ORDER BY NAME, SALARY;

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
3	Ajeet bhargav	45	Meerut	65000
5	Balwant singh	45	Varanasi	36000
1	Himani gupta	21	Modinagar	22000
6	Mahesh sharma	26	Mathura	22000
4	Ritesh yadav	36	Azamgarh	26000
2	Shiva tiwari	22	Bhopal	21000

This is an example to sort the result in descending order by NAME.

❖ **SELECT** * **FROM** CUSTOMERS
ORDER BY NAME DESC;

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
2	Shiva tiwari	22	Bhopal	21000
4	Ritesh yadav	36	Azamgarh	26000
6	Mahesh sharma	26	Mathura	22000
1	Himani gupta	21	Modinagar	22000
5	Balwant singh	45	Varanasi	36000
3	Ajeet bhargav	45	Meerut	65000

UNIT – 6

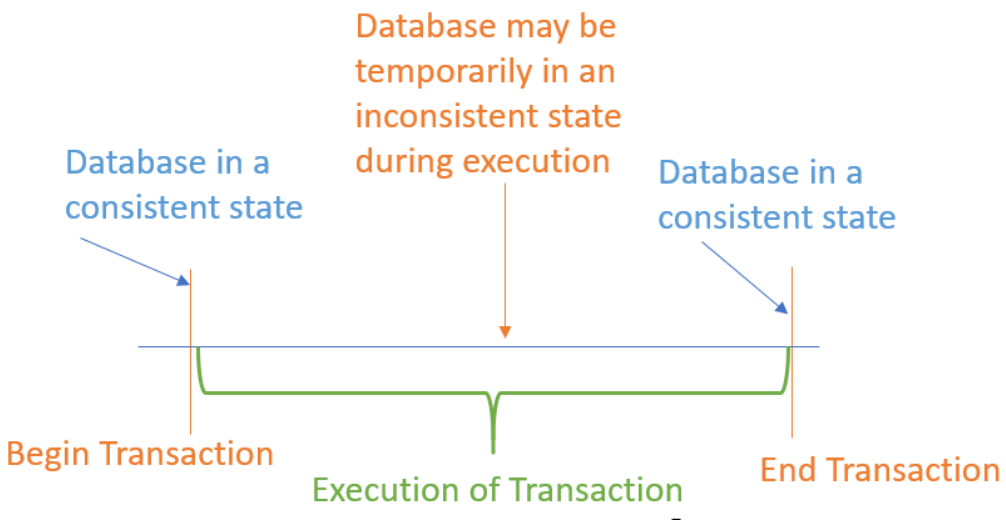
TRANSACTION PROCESSING CONCEPT

Transaction processing systems

- **Transaction processing systems** are systems with large databases and hundreds of concurrent users that are executing database transactions.
- Examples of such systems include systems for reservations, banking, credit card processing, stock markets, supermarket checkout, and other similar systems.
- They require high availability and fast response time for hundreds of concurrent users.

Transaction

- A **transaction** is a logical unit of database processing that includes one or more database access operations. These can include insertion, deletion, modification, or retrieval operations.
- The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.
- Transaction boundaries is by specifying explicit **begin transaction** and **end transaction** statements in an application program;
- In this case, all database access operations between the two are considered as forming one transaction.
- During the transaction the database is inconsistent.
- Only once the database is committed the state is changed from one consistent state to another.



- A transaction is a program unit whose execution may or may not change the contents of a database.
- The transaction concept in DBMS 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.

Example –

Transaction to be performed to withdraw cash from an ATM

Set of Operations :

Consider the following example for transaction operations as follows.

Example -ATM transaction steps.

- Transaction Start.
- Insert your ATM card.
- Select language for your transaction.
- Select Savings Account option.
- Enter the amount you want to withdraw.
- Enter your secret pin.

- Wait for some time for processing.
- Collect your Cash.
- Trasaction Completed.

Desirable Properties of Transactions

Transactions should possess several properties. These are often called the **ACID properties,** and they should be enforced by the concurrency control and recovery methods of the DBMS. ACID in DBMS stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability. The following are the ACID properties:

1. **Atomicity:**

   A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

2. **Consistency preservation:**

   A transaction is consistency preserving if its complete execution take(s) the database from one consistent state to another.

3. **Isolation:**

   A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

4. **Durability or permanency:**

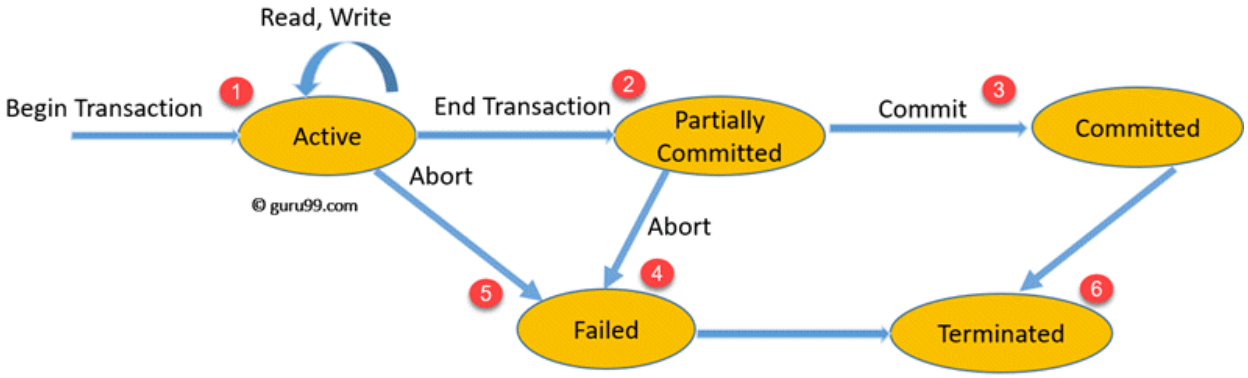
   The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

States of Transactions

The various states of a transaction concept in DBMS are :

State	Transaction types
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.



State Transition Diagram for a Database Transaction

Let's study a state transition diagram that highlights how a transaction moves between these various states.

1. Once a transaction states execution, it becomes **active**. It can issue READ or WRITE operation.
2. Once the READ and WRITE operations complete, the transactions becomes **partially committed** state.
3. Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the **committed** state.
4. If the check is a fail, the transaction goes to the **Failed** state.
5. If the transaction is aborted while it's in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
6. The **terminated** state refers to the transaction leaving the system.

Transaction Operation

- A **database** is basically represented as a collection of **named data items.**
- The size of a data item is called its **granularity,** and it can be a field of some record in the database, or it may be a larger unit such as a record or even a whole disk block.
- **But the concepts we discuss are independent of the data item granularity.**
- Using this simplified database model, the basic database access operations that a transaction can include are as follows:
    - ✓ **read_item(*X*):** Reads a database item named *X* into a program variable. To simplify our notation, we assume that *the program variable is also named X.*
    - ✓ **write_item(*X*):** Writes the value of program variable *X* into the database item named *X.*
- The basic unit of data transfer from disk to main memory is one block.
- Executing a **read_item(*X*)** command includes the following steps:
    1. Find the address of the disk block that contains item *X.*
    2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
    3. Copy item *X* from the buffer to the program variable named *X.*
- Executing a **write_item(*X*)** command includes the following steps:
    1. Find the address of the disk block that contains item *X.*
    2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
    3. Copy item *X* from the program variable named *X* into its correct location in the buffer.
    4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Example:-

Before x :700	Y:200
Transaction T	
T1	T2
Read(x) x: x-100 Write(x)	Read(y) y: y+100 Write(y)
After x :600	y:300

Why do you need 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.
- Not managing concurrent access may create issues like:
    - ✓ Hardware failure and system crashes
    - ✓ Concurrent execution of the same transaction, deadlock, or slow performance
- Concurrency control and recovery mechanisms are mainly concerned with the database access commands in a transaction.
- Transactions submitted by the various users may execute concurrently and may access and update the same database items.
- If this concurrent execution is uncontrolled, it may lead to problems, such as an inconsistent database.
- If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency exists.
- However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur.
- Here we discuss three of the problems that may occur are:
    - ✓ The Lost Update Problem
    - ✓ The Temporary Update (or Dirty Read) Problem
    - ✓ The Incorrect Summary Problem

The lost update problem

- The lost update problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
- That is, interleaved use of the same data item would cause some problems when an update operation from one transaction overwrites another update from a second transaction.

Example:-

- Suppose the two transactions T1 and T2 introduced previously are submitted at approximately the same time. It is possible when two travel agency staff help customers to book their flights at more or less the same time from a different or the same office. Suppose that their operations are interleaved by the operating system as shown in the figure below:

Time	T_1	T_2	Comments
1	read_item(X);		Read operation is performed in T_1 at time step 1.
2	X:=X-N;		Value of data item X is modified.
3		read_item(X);	Read in value of X (Which value is read in?)
4		X:=X+M	Value of data item X is modified.
5	write_item(X);		Write operation is performed in T_1.
6	read_item(Y);		Read operation is performed in T_1.
7		write_item(X);	Write data item X to database (What value is written in?)
8	Y:=Y+N;		Value of data item is modified.
9	write_item(Y);		Write data item Y to database.

- The above interleaved operation will lead to an incorrect value for data item X, because at time step 3, T2 reads in the original value of X which is before T1 changes it in the database, and hence the updated value resulting from T1 is lost.
- For example, if X = 80, originally there were 80 reservations on the flight, N = 5, T1 cancels 5 seats on the flight corresponding to X and reserves them on the flight corresponding to Y, and M = 4, T2 reserves 4 seats on X.

- The final result should be X = 80 - 5 + 4 = 79; but in the concurrent operations of the figure above, it is X = 84 because the update that cancelled 5 seats in T1 was lost.
- The detailed value updating in the flight reservation database in the above example is shown below:

Time	T_1	T_2	Value
1	read_item(X);		X = 80
2	X:=X-N;		X =80–5=75 (which is not written into database)
3		read_item(X);	X = 80 (T2 still reads in the original value of X, the updated value of X is lost.)
4		X:=X+M	X=80+4=84
5	write_item(X);		X=75 is written into database
6	read_item(Y);		
7		write_item(X);	X=84 over writes X=75, a wrong record is written in database
8	Y:=Y+N;		
9	write_item(Y);		

Uncommitted dependency (or dirty read / temporary update)

- Uncommitted dependency occurs when a transaction is allowed to retrieve or (worse) update a record that has been updated by another transaction, but which has not yet been committed by that other transaction.
- Because it has not yet been committed, there is always a possibility that it will never be committed but rather rolled back, in which case, the first transaction will have used some data that is now incorrect - a dirty read for the first transaction.

Example:-

- The figure below shows an example where T1 updates item X and then fails before completion, so the system must change X back to its original value.
- Before it can do so, however, transaction T2 reads the 'temporary' value of X, which will not be recorded permanently in the database because of the failure of T1.
- The value of item X that is read by T2 is called dirty data, because it has been created by a transaction that has not been completed and committed yet; hence this problem is also known as the dirty read problem.

- Since the dirty data read in by T2 is only a temporary value of X, the problem is sometimes called temporary update too.

Time	T_1	T_2	Comment
1	read_item(X);		
2	X:=X-N;		
3	write_item(X);		X is temporarily updated
4		read_item(X);	
5		X:=X+M	
6		write_item(X);	
7	read_item(Y);		
...	...	...	
	ROLLBACK		**T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the temporary incorrect value of X**

- The rollback of transaction T1 may be due to a system crash, and transaction T2 may already have terminated by that time, in which case the crash would not cause a rollback to be issued for T2. The following situation is even more unacceptable:

Time	T_1	T_2	Comment
1	read_item(X);		
2	X:=X-N;		
3	write_item(X);		X is temporarily updated
4		read_item(X);	
5		X:=X+M	
6		write_item(X);	
7		**COMMIT**	**T_2 loses update chance**
8	read_item(Y);		
...	...	...	
	ROLLBACK		**T_2 depends on uncommitted value and loses an update at time step 7**

- In the above example, not only does transaction T2 becomes dependent on an uncommitted change at time step 6, but it also loses an update at time step 7, because the rollback in T1 causes data item X to be restored to its value before time step 1.

The Incorrect Summary Problem (Inconsistent analysis)

- Inconsistent analysis occurs when a transaction reads several values, but a second transaction updates some of these values during the execution of the first.

- This problem is significant, for example, if one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records.
- The aggregate function may calculate some values before they are updated and others after they are updated. This causes an inconsistency.

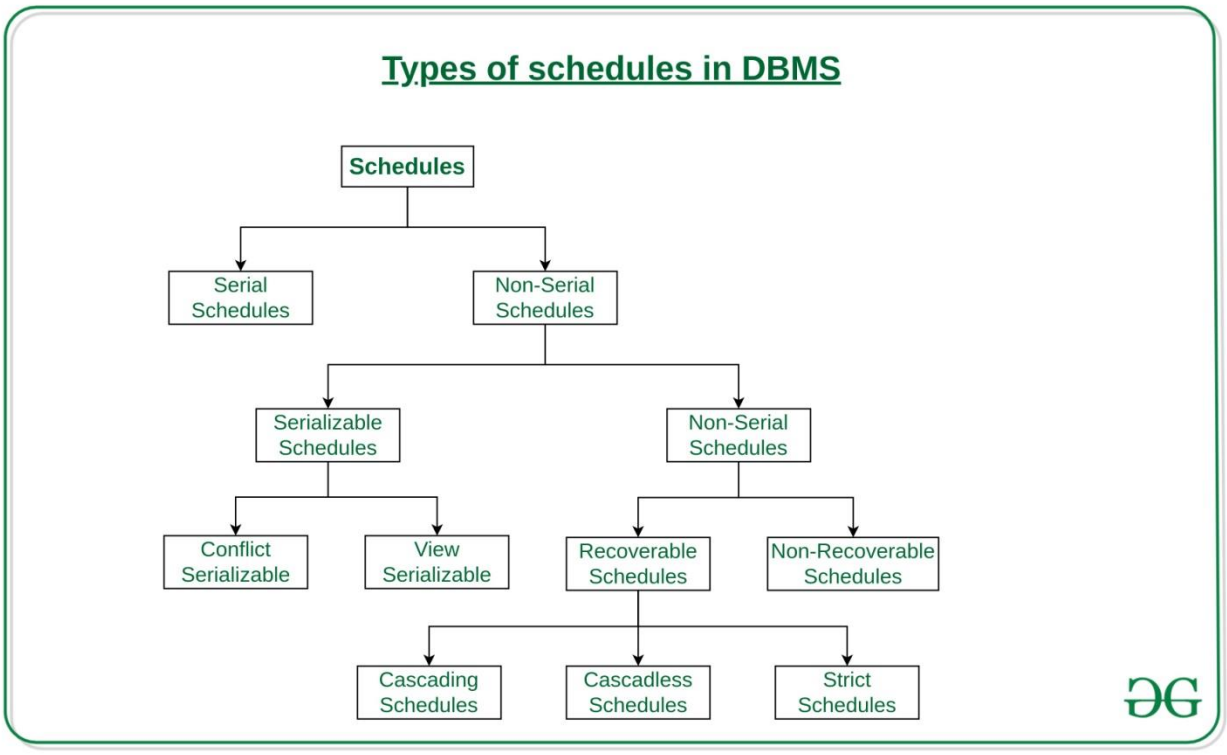
Example:-

- For example, suppose that a transaction T3 is calculating the total number of reservations on all the flights; meanwhile, transaction T1 is executing.
- If the interleaving of operations shown below occurs, the result of T3 will be off by amount N, because T3 reads the value of X after N seats have been subtracted from it, but reads the value of Y before those N seats have been added to it.

Time	T_1	T_3	Comment
1		sum:=0;	
2		read_item(A);	
3		sum:=sum+A;	
4	read_item(X);		
5	X:=X-N;		
6	write_item(X);		
7		read_item(X);	**T_3 reads X after N is subtracted from it**
8		sum:=sum+X;	
9		read_item(Y);	**T_3 reads Y before N is added to it**
10		sum:=sum+Y;	**A wrong summary is resulted (off by N)**
11	read_item(Y);		
12	Y:=Y+N;		
13	write_item(Y);		

Schedules

- Schedule is a process of lining the transactions and executing them one by one. When there are multiple transactions that are running in a concurrent manner and the order of operation is needed to be set so that the operations do not overlap each other, Scheduling is brought into play and the transactions are timed accordingly.



1. **Serial Schedules:**

- Schedules in which the transactions are executed non-interleaved, i.e., a serial schedule is one in which no transaction starts until a running transaction has ended are called serial schedules.

Example: Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
W(A)	
R(B)	
	W(B)
	R(A)
	R(B)

where R(A) denotes that a read operation is performed on some data item ‘A’. This is a serial schedule since the transactions perform serially in the order T_1 —> T_2

2. **Non-Serial Schedule:**

- This is a type of Scheduling where the operations of multiple transactions are interleaved. This might lead to a rise in the concurrency problem.
- The transactions are executed in a non-serial manner, keeping the end result correct and same as the serial schedule.
- Unlike the serial schedule where one transaction must wait for another to complete all its operation, in the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete.
- This sort of schedule does not provide any benefit of the concurrent transaction. It can be of two types namely, Serializable and Non-Serializable Schedule.

- The Non-Serial Schedule can be divided further into
    - ✓ Serializable schedule
    - ✓ Non-Serializable. schedule

Serializable schedule:

- This is used to maintain the consistency of the database. It is mainly used in the Non-Serial scheduling to verify whether the scheduling will lead to any inconsistency or not.
- On the other hand, a serial schedule does not need the serializability because it follows a transaction only when the previous transaction is complete.
- The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions.
- Since concurrency is allowed in this case thus, multiple transactions can execute concurrently.
- A serializable schedule helps in improving both resource utilization and CPU throughput.
- These are of two types:
    - ✓ Conflict Serializable

- ✓ View Serializable

Conflict Serializable schedule:

- A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations are said to be conflicting if all conditions satisfy:
  - They belong to different transactions
  - They operate on the same data item
  - At Least one of them is a write operation

View Serializable schedule:

- A Schedule is called view serializable if it is view equal to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind writes, then the view serializable does not conflict serializable.

Non-Serializable schedule:

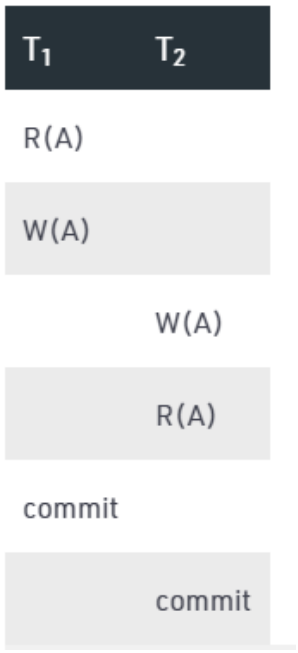
- The non-serializable schedule is divided into two types
  - ✓ Recoverable schedule
  - ✓ Non-recoverable Schedule.

Recoverable Schedule:

- Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules.
- In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.

Example –

Consider the following schedule involving two transactions T_1 and T_2.



- This is a recoverable schedule since T_1 commits before T_2, that makes the value read by T_2 correct.
- There can be three types of recoverable schedule:
    - ✓ Cascading Schedule:
    - ✓ Cascade less Schedule:
    - ✓ Strict Schedule:

Cascading Schedule:

- Cascading schedule also called Avoids cascading aborts/rollbacks (ACA).
- When there is a failure in one transaction and this leads to the rolling back or aborting other dependent transactions, then such scheduling is referred to as Cascading rollback or cascading abort.

Example:

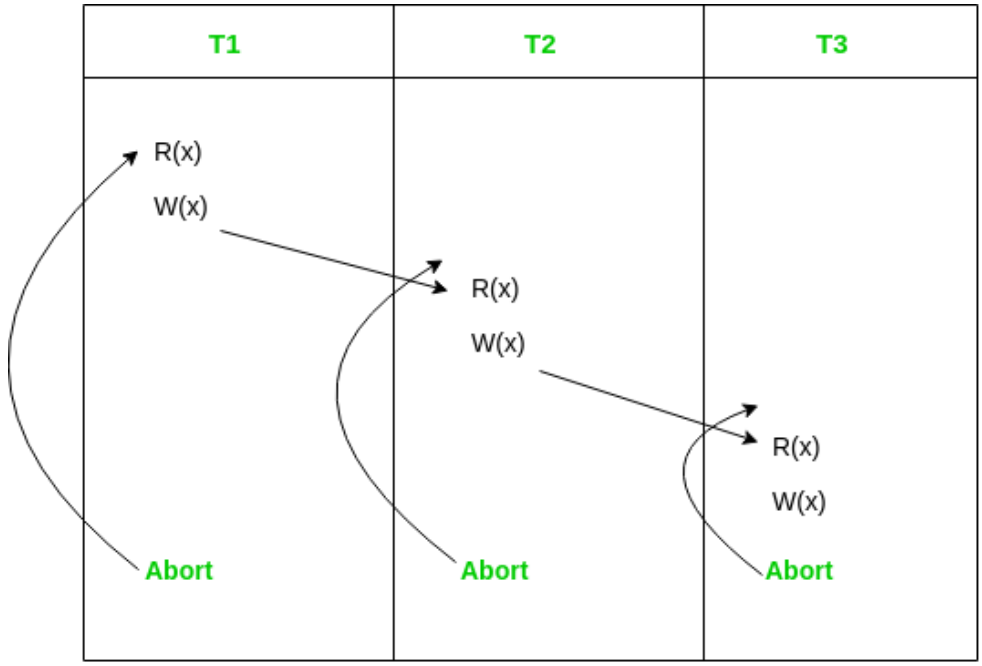


Figure - Cascading Abort

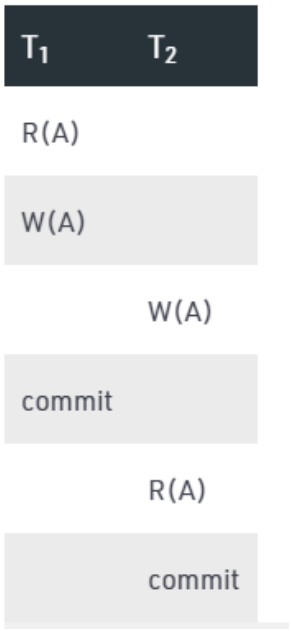
Cascade less Schedule:

- Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules.
- Avoids that a single transaction abort leads to a series of transaction rollbacks.
- A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

- In other words, if some transaction T_j wants to read value updated or written by some other transaction T_i, then the commit of T_j must read it after the commit of T_i.

Example:

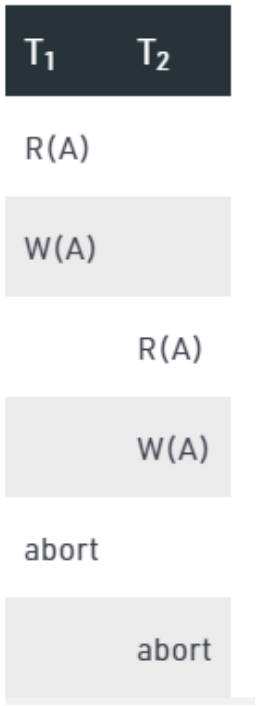
Consider the following schedule involving two transactions T_1 and T_2.



- This schedule is cascade less. Since the updated value of **A** is read by T_2 only after the updating transaction i.e. T_1 commits.

Example:

Consider the following schedule involving two transactions T_1 and T_2.



- It is a recoverable schedule but it does not avoid cascading aborts. It can be seen that if T_1 aborts, T_2 will have to be aborted too in order to maintain the correctness of the schedule as T_2 has already read the uncommitted value written by T_1.

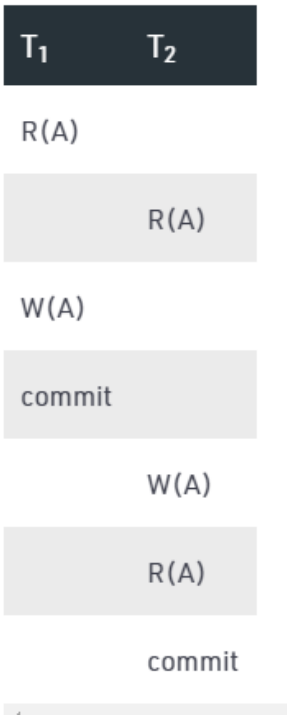
Strict Schedule:

- A schedule is strict if for any two transactions T_i, T_j, if a write operation of T_i precedes a conflicting operation of T_j (either read or write), then the commit or abort event of T_i also precedes that conflicting operation of T_j.

- In other words, T_j can read or write updated or written value of T_i only after T_i commits/aborts.

Example:

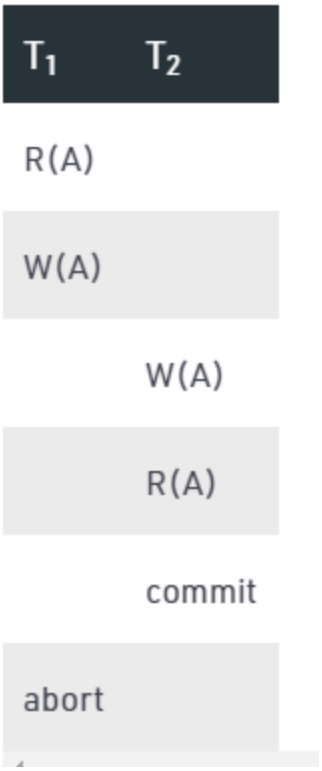
Consider the following schedule involving two transactions T_1 and T_2.



- This is a strict schedule since T_2 reads and writes A which is written by T_1 only after the commit of T_1.

Non-Recoverable Schedule:

Example: Consider the following schedule involving two transactions T_1 and T_2.

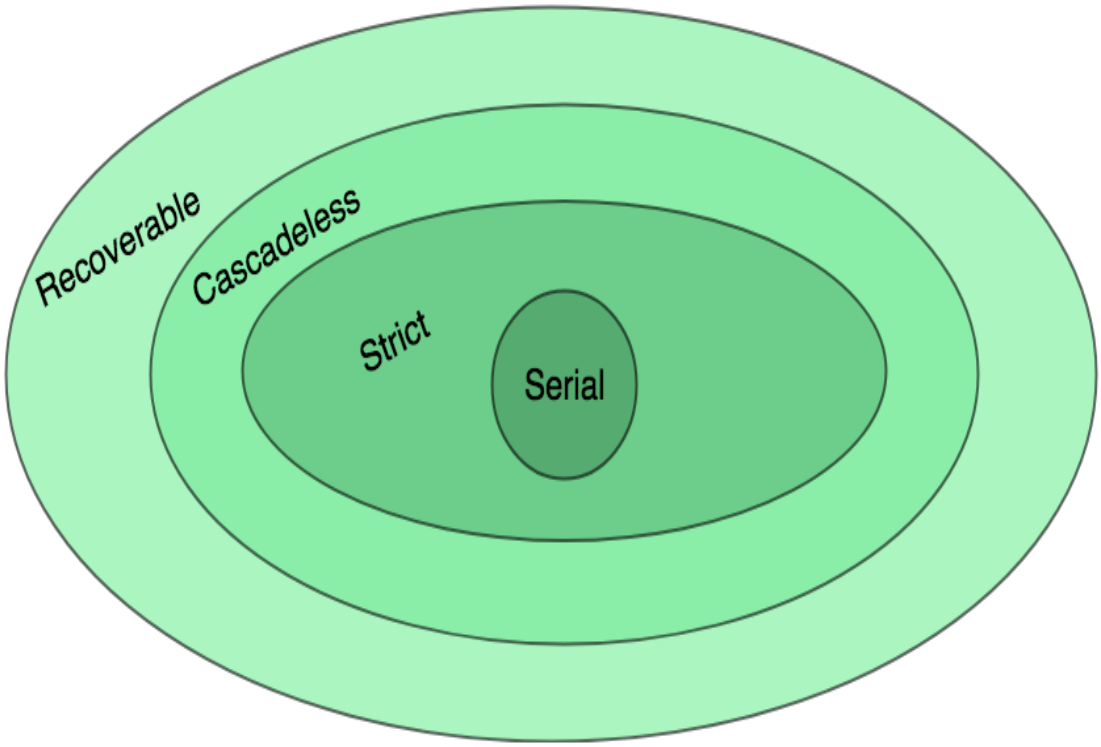


- T_2 read the value of A written by T_1, and committed. T_1 later aborted, therefore the value read by T_2 is wrong, but since T_2 committed, this schedule is **non-recoverable**.

Note – It can be seen that:

1. Cascadeless schedules are stricter than recoverable schedules or are a subset of recoverable schedules.
2. Strict schedules are stricter than cascadeless schedules or are a subset of cascadeless schedules.
3. Serial schedules satisfy constraints of all recoverable, cascadeless and strict schedules and hence is a subset of strict schedules.

The relation between various types of schedules can be depicted as:



Example:

Consider the following schedule:

S:R1(A), W2(A), Commit2, W1(A), W3(A), Commit3, Commit1

Which of the following is true?

(A) The schedule is view serializable schedule and strict recoverable schedule

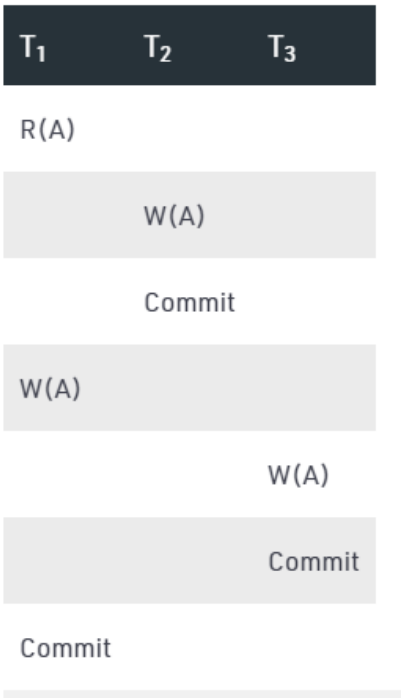
(B) The schedule is non-serializable schedule and strict recoverable schedule

(C) The schedule is non-serializable schedule and is not strict recoverable schedule.

(D) The Schedule is serializable schedule and is not strict recoverable schedule

Solution:

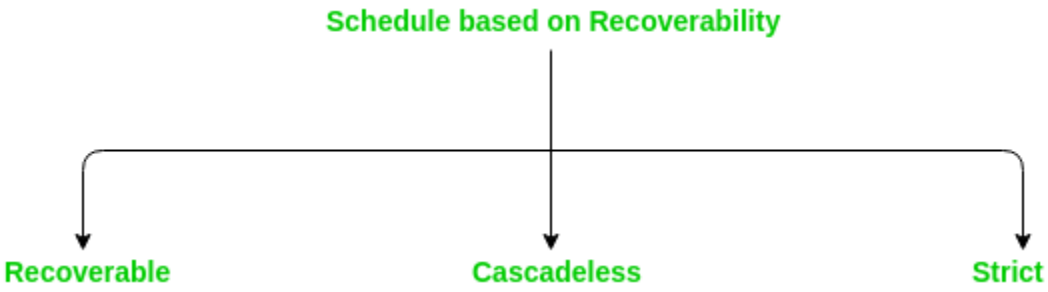
The schedule can be re-written as:-



- First of all, it is a view serializable schedule as it has view equal serial schedule T_1 —> T_2 —> T_3 which satisfies the initial and updated reads and final write on variable A which is required for view serializability.
- Now we can see there is write – write pair done by transactions T_1 followed by T_3 which is violating the above-mentioned condition of strict schedules as T_3 is supposed to do write operation only after T_1 commits which is violated in the given schedule.
- Hence the given schedule is serializable but not strict recoverable. So, option (D) is correct.

Types of Schedules based Recoverability in DBMS

Generally, there are three types of schedule given as follows:



1. **Recoverable Schedule –**

- A schedule is said to be recoverable if it is recoverable as name suggest. Only reads are allowed before write operation on same data. Only reads (Ti->Tj) is permissible.

Example –

S1: R1(x), **W1(x)**, R2(x), R1(y), R2(y),
W2(x), W1(y), **C1**, **C2**;

- Given schedule follows order of **Ti->Tj => C1->C2**. Transaction T1 is executed before T2 hence there is no chances of conflict occur. R1(x) appears before W1(x) and transaction T1 is committed before T2 i.e. completion of first transaction performed first update on data item x, hence given schedule is recoverable.

- Let's see example of **unrecoverable schedule** to clear the concept more:

Example –

S2: R1(x), R2(x), R1(z), R3(x), R3(y), W1(x),
W3(y), R2(y), W2(z), **W2(y)**, C1, **C2**, **C3**;

- **Ti->Tj => C2->C3** but W3(y) executed before W2(y) which leads to conflicts thus it must be committed before T2 transaction. So given schedule is unrecoverable. if **Ti->Tj => C3->C2** is given in schedule then it will become recoverable schedule.

Note: A committed transaction should never be rollback. It means that reading value from uncommitted transaction and commit it will enter the current transaction into inconsistent or unrecoverable state this is called *Dirty Read problem*.

Example:

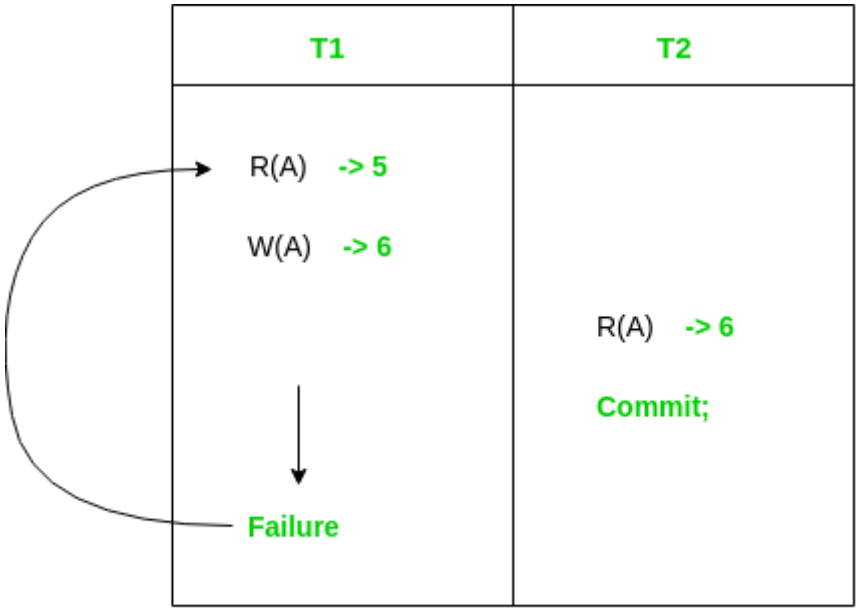


Figure - Dirty Read Problem

2. **Cascade less Schedule –**
    - When no **read** or **write-write** occurs before execution of transaction then corresponding schedule is called cascade less schedule.

Example –

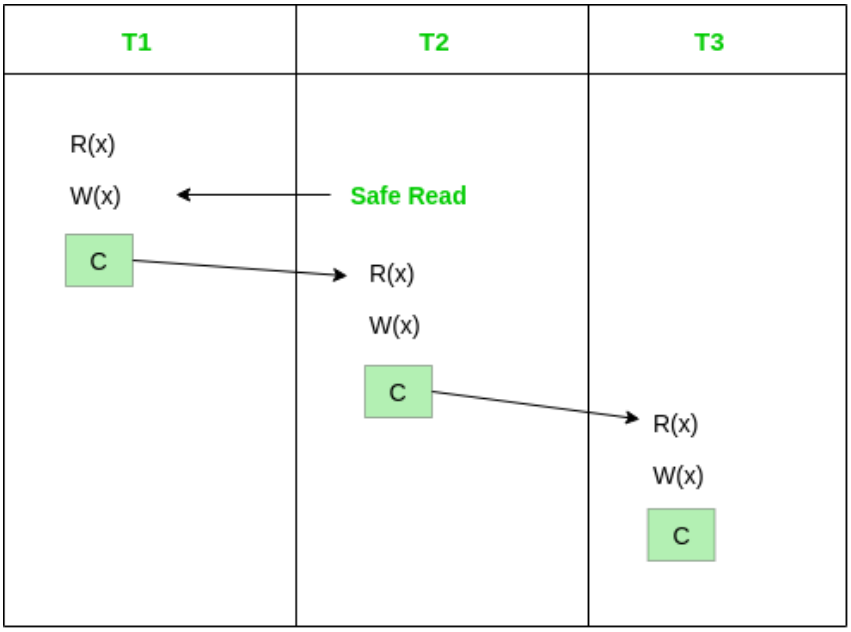
S3: R1(x), R2(z), R3(x), R1(z), R2(y), R3(y), W1(x), C1,

W2(z), **W3(y)**, **W2(y)**, **C3**, **C2**;

- In this schedule **W3(y)** and **W2(y)** overwrite conflicts and there is no read, therefore given schedule is cascade less schedule.

Special Case –

- A committed transaction desired to abort. As given below all the transactions are reading committed data hence it's cascadeless schedule.



3. **Strict Schedule –**
    - If schedule contains no **read** or **write** before commit then it is known as strict schedule. Strict schedule is strict in nature.

Example –

S4: R1(x), R2(x), R1(z), R3(x), R3(y),

W1(x), **C1**, **W3(y)**, **C3**, R2(y), **W2(z)**, **W2(y)**, **C2**;

- In this schedule no read-write or write-write conflict arises before commit hence its strict schedule:

T1	T2	T3
R1(x) R1(z) **W1(x)** **C1;**	R2(x) R2(y) **W2(z)** **W2(y)** **C2;**	 R3(x) R3(y) **W3(y)** **C3;**

Figure - Strict Schedule

4. **Cascading Abort –**

- Cascading Abort can also be rollback. If transaction T1 abort as T2 read data that written by T1 which is not committed. Hence it's cascading rollback.

Example:

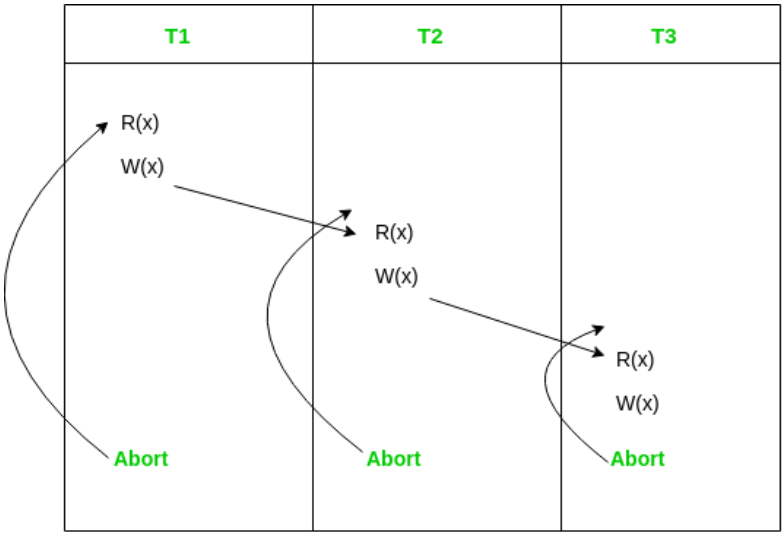


Figure - Cascading Abort

Why Recovery Is Needed

- Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either

  (1) All the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or

  (2) The transaction has no effect whatsoever on the database or on any other transactions.

- The DBMS must not permit some operations of a transaction *T* to be applied to the database while other operations of *T* are not.
- This may happen if a transaction **fails** after executing some of its operations but before executing all of them.

Failure Classification

To find that where the problem has occurred, we generalize a failure into the following categories:

1. Transaction failure
2. System crash
3. Disk failure

1. Transaction failure

- The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

1. **Logical errors:** If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs.
2. **Syntax error:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For**

example, The system aborts an active transaction, in case of deadlock or resource unavailability.

2. System Crash

- System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.
- **Fail-stop assumption:** In the system crash, non-volatile storage is assumed not to be corrupted.

3. Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

UNIT – 7

CONCURRENCY CONTROL CONCEPT

CONCURRENCY CONTROL

- **Concurrency Control** in Database Management System is a procedure of managing simultaneous operations without conflicting with each other.
- It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.
- Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge.
- DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system.
- Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

Potential problems of Concurrency

Some issues which you will likely to face while using the DBMS Concurrency Control method:

- **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
- Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
- **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
- **Incorrect Summary issue** occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction updates few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

Why use Concurrency method?

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Example

Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:

- Lock-Based Protocols
- Two Phase Locking Protocol
- Timestamp-Based Protocols
- Validation-Based Protocols

Lock-based Protocols

- **Lock Based Protocols** in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock.

- Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a single user.

Lock

- A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks in DBMS help synchronize access to the database items by concurrent transactions.
- All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

Types of Lock

❖ Binary Locks:

- A **binary lock** can have two **states** or **values:** locked and unlocked or (1 and 0).
- A distinct lock is associated with each database item *X*.
- If the value of the lock on *X* is 1, item *X cannot be accessed* by a database operation that requests the item.
- If the value of the lock on *X* is 0, the item can be accessed when requested.
- Two operations, lock_item and unlock_item, are used with binary locking.
- A transaction requests access to an item *X* by first issuing a **lock_item(*X*)** operation.
- If LOCK(*X*) = 1, the transaction is forced to wait. If LOCK(*X*) = 0, it is set to 1 (the transaction **locks** the item) and the transaction is allowed to access item *X*.
- When the transaction is through using the item, it issues an **unlock_item(*X*)** operation, which sets LOCK(*X*) to 0 (**unlocks** the item) so that *X* may be accessed by other transactions.

The system needs to maintain only these records for the items that are currently locked in a **lock table,** which could be organized as a hash file. Items not in the lock table are considered to be unlocked. The DBMS has a **lock manager subsystem** to keep track of and control access to locks.

To use the binary locking scheme every transaction must obey the following rules:

1. A transaction T must issue the operation lock_item(*X*) before any read_item(*X*) or write_item(*X*) operations are performed in T.
2. A transaction T must issue the operation unlock_item(*X*) after all read_item(*X*) and write_item(*X*) operations are completed in T.
3. A transaction T will not issue a lock_item(*X*) operation if it already holds the lock on item *X* (Note 1).
4. A transaction T will not issue an unlock_item(*X*) operation unless it already holds the lock on item *X.*

❖ Shared / Exclusive:

This type of locking mechanism separates the locks in DBMS based on their uses.

❖ Shared Lock (S):

- A **shared lock** is also called a **Read-only lock**. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.

Example

Consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

❖ Exclusive Lock (X):

- With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

Example

When a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

When we use the shared/exclusive locking scheme, the system must enforce the following rules:

1. A transaction T must issue the operation read_lock(*X*) or write_lock(*X*) before any read_item(*X*) operation is performed in T.
2. A transaction T must issue the operation write_lock(*X*) before any write_item(*X*) operation is performed in T.
3. A transaction T must issue the operation unlock(*X*) after all read_item(*X*) and write_item(*X*) operations are completed in T .
4. A transaction T will not issue a read_lock(*X*) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.
5. A transaction T will not issue a write_lock(*X*) operation if it already holds a read (shared) lock or write (exclusive) lock on item X.
6. A transaction T will not issue an unlock(*X*) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item *X.*

❖ Conversion of Locks

- A transaction that already holds a lock on item *X* is allowed under certain conditions to **convert** the lock from one locked state to another.
- For example, it is possible for a transaction T to issue a read_lock(*X*) and then later on to **upgrade** the lock by issuing a write_lock(*X*) operation.
- If T is the only transaction holding a read lock on *X* at the time it issues the write_lock(*X*) operation, the lock can be upgraded; otherwise, the transaction must wait.
- It is also possible for a transaction T to issue a write_lock(*X*) and then later on to **downgrade** the lock by issuing a read_lock(*X*) operation.

- When upgrading and downgrading of locks is used, the lock table must include transaction identifiers in the record structure for each lock to store the information on which transactions hold locks on the item.

❖ Simplistic Lock Protocol

This type of lock-based protocols allows transactions to obtain a lock on every object before beginning operation. Transactions may unlock the data item after finishing the 'write' operation.

❖ Pre-claiming Locking

Pre-claiming lock protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process. In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

Starvation

- Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.
- Following are the reasons for Starvation:

  - ✓ When waiting scheme for locked items is not properly managed
  - ✓ In the case of resource leak
  - ✓ The same transaction is selected as a victim repeatedly

- When we use locking is **starvation,** which occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally.
- This may occur if the waiting scheme for locked items is unfair, giving priority to some transactions over others.
- One solution for starvation is to have a fair waiting scheme, such as using a **first-come-first-serve** queue; transactions are enabled to lock an item in the order in which they originally requested the lock.

- Another scheme allows some transactions to have priority over others but increases the priority of a transaction the longer it waits, until it eventually gets the highest priority and proceeds.

Deadlock

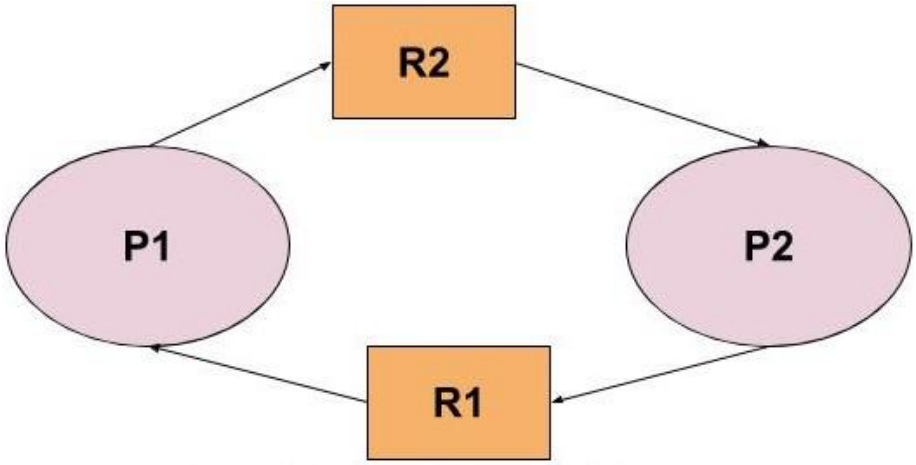
- Deadlock is an unwanted condition where two or more processes waits indefinitely for the other process to free up their resources so that they can complete their work.
- In Deadlock, one process is holding some resource and is wating for some other resource and at the same time, there is some other process that is holding that resource and is waiting for the resources held by the first process.
- So, both the processses are waiting for each other to release the resources and none of them are releasing the resources.

Example:

Suppose you are travelling on a one-way road by car. Suddenly you see a car approaching from the opposite direction. Now, you wait for the person to take a back turn and provide the road to you. But, the person in front of you is waiting for you to give him space. This is a deadlock situation where you both are stuck and nobody is ready to take back turn. Such type of situation may occur in our computer system also. Such a situation is called a deadlock.

Example:

- Consider a situation where there are two processes P1 and P2 executing simultaneously.
- The process P1 is holding the resource R1 and needs R2 to complete its execution.
- The process P2 is holding the resource R2 and needs R1 to complete its execution.
- So, here one process is waiting for the other process to release its resource and hence both the process is unable to complete its execution and the system goes into a halt.



Process P1 is holding R1 and waiting for R2 while Process P2 is holding R2 and waiting for R1.

- This is called the Deadlock situation. But there are some conditions that must be satisfied for Deadlock.

Necessary conditions for Deadlock

1. **Mutual Exclusion:** This means that some of the resources involved cannot be used by more than one process at the same time. It means that if the resource(like a printer) is being used by any process and some other process comes then it has to wait for the resources to be released.
2. **No Preemption:** If a process is holding a resource then its resources can not be forcibly taken from it.
3. **Hold & Wait:** The process is holding some resources and is waiting for the other resources.
4. **Circular Wait:** The process is waiting for the resource which is being held by the next process and this forms a chain of waiting. In the above example, Process P1 and Process P2 are forming a chain and waiting for the resource being held by each other.

- If the above four conditions hold simultaneously then a Deadlock may occur.

Deadlock Avoidance

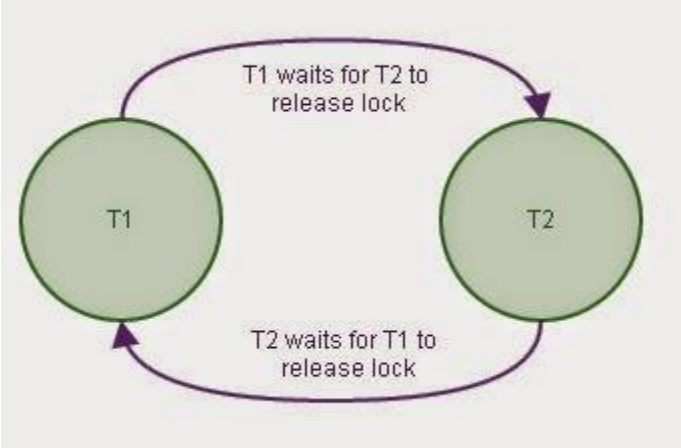
- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database.
- For the larger database, deadlock prevention method can be used.

Deadlock detection (Wait for Graph)

- The deadlock can be detected by the resource scheduler who checks all the resources allocated to the different processes.
- The deadlock should be avoided instead of terminating and restarting the transaction so that both resources as well time is not wasted.
- One of the methods for detecting deadlock is Wait-For-Graph which is suitable for smaller databases.

Wait-For-Graph

- A graph is created based on the transactions and locks on the resource in this method.
- A deadlock occurs if the graph which is created has a closed-loop. For all transactions waiting the resources are maintained by DBMS and also are checked to see if there is any closed loop.
- Let us consider two transactions T1 and T2 where T1 requests for a resource held by T2. The wait-for-graph in this scenario draws an arrow from T1 to T2 and when the resource is released by T2, the arrow gets deleted.
- For example, T1 requests for a lock X on a resource, which is held by T2, a directed edge is created from T1 to T2. When T2 releases the resource X, the edge T1 locks the resource and the directed edge between T1 and T2 is dropped.



Deadlock Prevention

- In DBMS, all the operations are analyzed and inspected by it to find if there is any possibility of occurrence of deadlock and in case of the possibility of deadlock, the transaction is not allowed to be processed.
- Primarily, the timestamp at which the transactions have begun is examined by the DBMS and based on this the transactions are ordered.
- The deadlock can be prevented by using schemes that use the timestamp of the transactions to calculate the occurrence of deadlock.

1. Wait- Die Scheme

- In this scheme, when a transaction requests for the resource which is already held by another transaction, then the timestamps of the transactions are scanned by the DBMS and the older transaction waits till the resource becomes available.

- Let us consider two transactions T1 and T2 and the timestamp of the transaction be denoted by TS.

- If T1 requests for resources held by T2 and a lock exists on T2 by some other transaction,

Below are the steps followed:

- Whether TS(T1) < TS(T2) is examined, and if T1 is the older transaction between T1 and T2 and some resource is held by the transaction T2, then it permits T1 to await the resource to be available for execution, otherwise abort T1 & restart it later with the same timestamp.
- If T1 is the older transaction that has held some resource and T2 is waiting for the resource held by T1, then T2 gets killed and later it will be restarted with the same timestamp but with a random delay.

2. Wound Wait Scheme

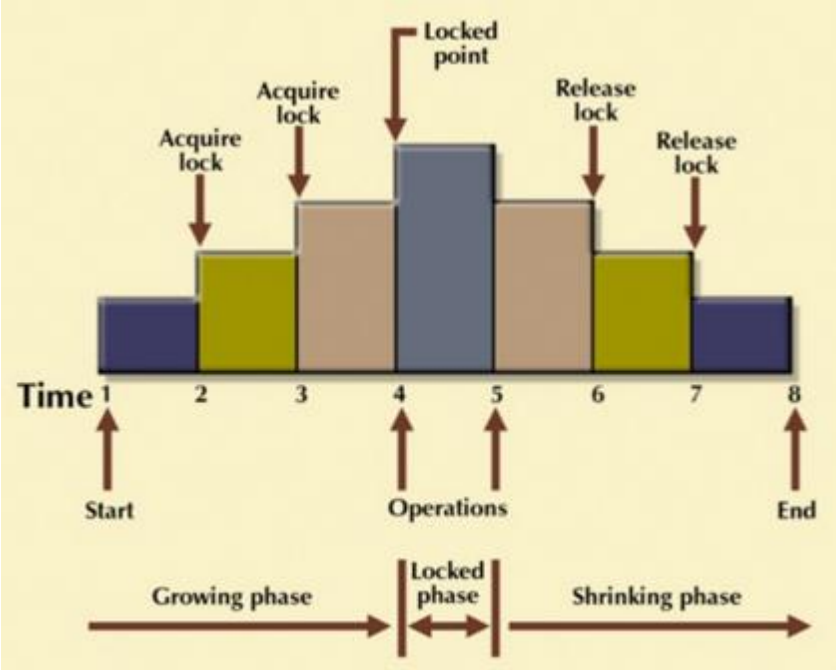
- In this scheme, if T1 is the older transaction in between transactions T1 and T2, and when T2 requests for the resource which is held by the transaction T1, then the younger transaction i.e. T2 waits until T1 releases the resource.

- But when the resource held by the younger transaction T2 is requested by the older transaction T1, T2 is forced by T1 to kill the transaction in order to release the resource held by it and afterward T2 is restarted with a delay but with the same timestamp.

Two Phase Locking Protocol

- **Two Phase Locking Protocol** also known as **2PL** protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously.
- Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.
- This locking protocol divides the execution phase of a transaction into three different parts.

    - ✓ In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
    - ✓ The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
    - ✓ In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.



- The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

    - ✓ **Growing Phase**: In this phase transaction may obtain new locks but may not release any locks.
    - ✓ **Shrinking Phase**: In this phase, a transaction may release existing locks but not obtain any new lock

- If lock conversion is allowed ,then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, & downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase.
- It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.
- In the above-given diagram, you can see that local and global deadlock detectors are searching for deadlocks and solve them with resuming transactions to their initial states.

Types of two phase locking (2PL)

- Basic Two-Phase Locking
- Conservative Two-Phase Locking

- Strict Two-Phase Locking
- Rigorous Two-Phase Locking

❖ Basic Two-Phase Locking

The technique just described above is known as **basic 2PL.**

❖ Conservative Two-Phase Locking

- A variation known as **conservative 2PL** (or **static 2PL)** requires a transaction to lock all the items it accesses *before the transaction begins execution,* by **predeclaring** its *read-set* and *write-set.*
- The **read-set** of a transaction is the set of all items that the transaction reads, and the **write-set** is the set of all items that it writes.
- If any of the predeclared items needed cannot be locked, the transaction does not lock any item; instead, it waits until all the items are available for locking.
- Conservative 2PL is a deadlock-free protocol.

❖ Strict Two-Phase Locking

- The most popular variation of 2PL is **strict 2PL,** which guarantees strict schedules .
- In this variation, a transaction T does not release any of its exclusive (write) locks until after it commits or aborts.
- Hence, no other transaction can read or write an item that is written by T unless T has committed, leading to a strict schedule for recoverability.
- Strict 2PL is not deadlock-free.

❖ Rigorous Two-Phase Locking

- A more restrictive variation of strict 2PL is **rigorous 2PL,** which also guarantees strict schedules.
- In this variation, a transaction T does not release any of its locks (exclusive or shared) until after it commits or aborts, and so it is easier to implement than strict 2PL.

- The difference between conservative and rigorous 2PL; the former must lock all its items *before it starts* so once the transaction starts it is in its shrinking phase, whereas the latter does not unlock any of its items until *after it*

Timestamp-based Protocols

- **Timestamp based Protocol** in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions.
- The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.
- The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.
- Lock-based protocols help you to manage the order between the conflicting transactions when they will execute.
- Timestamp-based protocols manage conflicts as soon as an operation is created.

Example:

```
Suppose there are there transactions T1, T2, and T3.
T1 has entered the system at time 0010
T2 has entered the system at 0020
T3 has entered the system at 0030
Priority will be given to transaction T1, then transaction T2 and lastly Transaction
T3.
```

Advantages:

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:

- Starvation is possible if the same transaction is restarted and continually aborted

Validation Based Protocol

- **Validation based Protocol** in DBMS also known as **Optimistic Concurrency Control Technique** is a method to avoid concurrency in transactions.
- In this protocol, the local copies of the transaction data are updated rather than the data itself, which results in less interference while execution of the transaction.
- The Validation based Protocol is performed in the following three phases:

    1. Read Phase
    2. Validation Phase
    3. Write Phase

Read Phase

In the Read Phase, the data values from the database can be read by a transaction but the write operation or updates are only applied to the local data copies, not the actual database.

Validation Phase

In Validation Phase, the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.

Write Phase

In the Write Phase, the updates are applied to the database if the validation is successful, else; the updates are not applied, and the transaction is rolled back.

Characteristics of Good Concurrency Protocol

An ideal concurrency control DBMS mechanism has the following objectives:

- Must be resilient to site and communication failures.
- It allows the parallel execution of transactions to achieve maximum concurrency.

- Its storage mechanisms and computational methods should be modest to minimize overhead.
- It must enforce some constraints on the structure of atomic actions of transactions.

Summary

- Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another.
- Lost Updates, dirty read, Non-Repeatable Read, and Incorrect Summary Issue are problems faced due to lack of concurrency control.
- Lock-Based, Two-Phase, Timestamp-Based, Validation-Based are types of Concurrency handling protocols
- The lock could be Shared (S) or Exclusive (X)
- Two-Phase locking protocol which is also known as a 2PL protocol needs transaction should acquire a lock after it releases one of its locks. It has 2 phases growing and shrinking.
- The timestamp-based algorithm uses a timestamp to serialize the execution of concurrent transactions. The protocol uses the **System Time or Logical Count as** a Timestamp.

Serializability

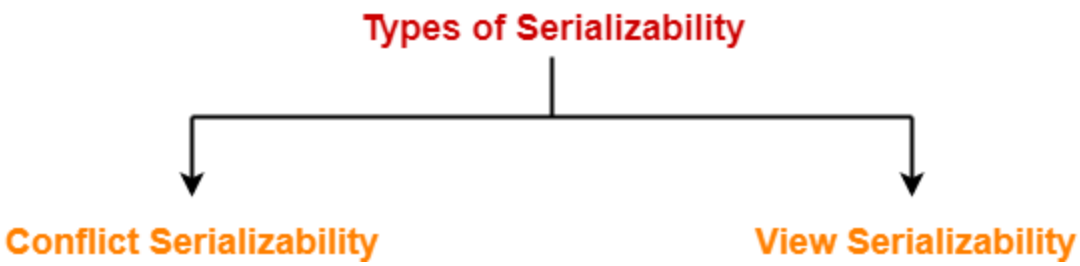
- Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order.
- It assumes that all accesses to the database are done using read and write operations.
- A schedule is called ``correct'' if we can find a serial schedule that is ``equivalent'' to it.
- Given a set of transactions T1...Tn, two schedules S1 and S2 of these transactions are equivalent if the following conditions are satisfied:

    - ✓ ***Read-Write Synchronization*****:** If a transaction reads a value written by another transaction in one schedule, then it also does so in the other schedule.
    - ✓ ***Write-Write Synchronization*****:** If a transaction overwrites the value of another transaction in one schedule, it also does so in the other schedule.

- Suppose that two users—two airline reservation clerks—submit to the DBMS transactions at approximately the same time. If no interleaving of operations is permitted, there are only two possible outcomes:
    - Execute all the operations of transaction (in sequence) followed by all the operations of transaction (in sequence).
    - Execute all the operations of transaction (in sequence) followed by all the operations of transaction (in sequence).

- If interleaving of operations is allowed, there will be many possible orders in which the system can execute the individual operations of the transactions.
- The concept of **serializability of schedules** is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules.

Types of Serializability-



1. Conflict Serializability
2. View Serializability

Conflict Serializability-

- If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

Conflicting Operations-

- Two operations are called as **conflicting operations** if all the following conditions hold true for them-

  - ✓ Both the operations belong to different transactions
  - ✓ Both the operations are on the same data item
  - ✓ At least one of the two operations is a write operation

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

❖ **Checking Whether a Schedule is Conflict Serializable Or Not-**

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01: Find and list all the conflicting operations.

Step-02: Start creating a precedence graph by drawing one node for each transaction.

Step-03: Draw an edge for each conflict pair such that if X_i (V) and Y_j (V) forms a conflict pair then draw an edge from T_i to T_j.

- ✓ This ensures that T_i gets executed before T_j.

Step-04: Check if there is any cycle formed in the graph.

- ✓ If there is no cycle found, then the schedule is conflict serializable otherwise not.

View Serializability-

- If a given schedule is found to be view equivalent to some serial schedule , then it is called as a view serializable schedule.

View Equivalent Schedules-

- Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.
- Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Condition-01:

- For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Thumb Rule : "Initial readers must be same for all the data items".

Condition-02:

- If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Thumb Rule : "Write-read sequence must be same.".

Condition-03:

- For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Thumb Rule : "Final writers must be same for all the data items".

❖ **Checking Whether a Schedule is View Serializable Or Not-**

Method-01:

Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be view serializable. Go and check using other methods.

Thumb Rules

- All conflict serializable schedules are view serializable.
- All view serializable schedules may or may not be conflict serializable.

Method-02:

Check if there exists any blind write operation. (Writing without reading is called as a blind write).

- If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.
- If there exists any blind write, then the schedule may or may not be view serializable. Go and check using other methods.

Thumb Rule

- No blind write means not a view serializable schedule.

Method-03:

In this method, try finding a view equivalent serial schedule.

- By using the above three conditions, write all the dependencies.
- Then, draw a graph using those dependencies.
- If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

UNIT-8 SECURITY & INTEGRITY

Authentication

- User authentication is to make sure that the person accessing the database is who he claims to be.
- Authentication can be done at the operating system level or even the database level itself.
- Many authentication systems such as retina scanners or bio-metrics are used to make sure unauthorized people cannot access the database.

Authorization

- Authorization is a privilege provided by the Database Administer.
- Users of the database can only view the contents they are authorized to view. The rest of the database is out of bounds to them.
- The different permissions for authorizations available are:

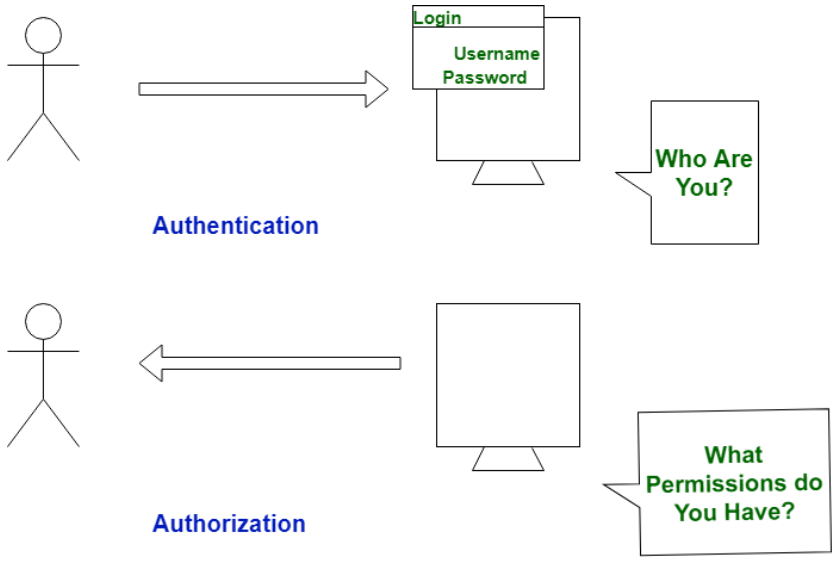
    1. **Primary Permission -** This is granted to users publicly and directly.
    2. **Secondary Permission -** This is granted to groups and automatically awarded to a user if he is a member of the group.
    3. **Public Permission -** This is publicly granted to all the users.
    4. **Context sensitive permission -** This is related to sensitive content and only granted to a select users.

- The categories of authorization that can be given to users are:

    1. **System Administrator -** This is the highest administrative authorization for a user. Users with this authorization can also execute some database administrator commands such as restore or upgrade a database.
    2. **System Control -** This is the highest control authorization for a user. This allows maintenance operations on the database but not direct access to data.
    3. **System Maintenance -** This is the lower level of system control authority. It also allows users to maintain the database but within a database manager instance.

4. **System Monitor -** Using this authority, the user can monitor the database and take snapshots of it.

- A user may have several forms of authorization on parts of the database. There are the following authorization rights.
    1. **Read authorization** allows reading, but not modification, of data.
    2. **Insert authorization** allows insertion of new data, but not modification of existing data.
    3. **Update authorization** allows modification, but not deletion of data.
    4. **Delete authorization** allows deletion of data.
- A user may be assigned all, none, ‘or a combination of these types of authorization.
- In addition to these forms of authorization for access to data, a user may be granted authorization to modify the database schema:
    1. **Index authorization** allows the creation and deletion of indexes.
    2. **Resource authorization** allows the creation of new relations.
    3. **Alteration authorization** allows the addition or deletion of attributes in a relation.
    4. **Drop authorization** allows the deletion of relations.
- The drop and delete authorization differ in that delete authorization allows deletion of tuples only.
- If a user deletes all tuples of a relation, the relation still exists, but it is empty.
- If a relation is dropped it no longer exists.
- The ability to create new relations is regulated through resource authorization.
- A user with resource authorization who creates a relation is given a privilege on that relation automatically.
- Index authorization is given to user to get the fast access of data on the bases of some key field.



VIEWS:-

- A *database view* is a subset of a database and is based on a query that runs on one or more database tables.
- Database views are saved in the database as named queries and can be used to save frequently used, complex queries.
- There are two types of database views:
    - ✓ dynamic views
    - ✓ static views.
- Dynamic views can contain data from one or two tables and automatically include all of the columns from the specified table or tables.
- Dynamic views are automatically updated when related objects or extended objects are created or changed.
- Static views can contain data from multiple tables and the required columns from these tables must be specified in the SELECT and WHERE clauses of the static view.
- Static views must be manually updated when related objects or extended objects are created or changed.
- It is possible to create views in SQL. A relation view is virtual since no corresponding physical relation exists. A view represents a different perspective of a base relation or relations.
- The result of a query operation on one or more base relations is a relation.

- Therefore if a user needs a particular view based on the base relations, it can be defined using a query expression. To be useful, we assign the view a name and relate it to the query expression.

Create view <view name> as <query expression>

- A view is a relation (virtual rather than base) and can be used in query an expression that is queries can be written using views as a relation.
- Views generally are not stored, since the data in the base relations may change.
- The definition of a view in a create view statement is stored in the system catalog. Having been defined, it can be used as if the view really represents a real relation. However such a virtual relation defined by a view is recomputed whenever a query refers to it.
- Views or sub schemes are used to enforce security. A user is allowed access to only that portion of the database defined by the user's view.
- A number of users may share a view. However, the users may create new views based on the views allowed.
- The advantage of this approach is that the number of objects accessible to a class of users and the entry for it in the authorization matrix is reduced one per view. This reduces the size of authorization matrix.
- The disadvantage is that the entire class of users has the same access rights.

- We can customize all aspects of a view, including:
  - ✓ The name of the view
  - ✓ The fields that appear in the view
  - ✓ The column title for each field in the view
  - ✓ The order of the fields in the view
  - ✓ The width of columns in the view, as well as the overall width of the view
  - ✓ The set of records that appear in the view (Filtering)
  - ✓ The order in which records are displayed in the view (Sorting & Grouping)
  - ✓ Column totals for numeric and currency fields (Totaling & Subtotaling)

Database Security

- Database security encompasses a range of security controls designed to protect the Database Management System (DBMS).
- The types of database security measures your business should use include protecting the underlying infrastructure that houses the database such as the network and servers), securely configuring the DBMS, and the access to the data itself.

Database security controls

- Database security encompasses multiple controls, including system hardening, access, DBMS configuration, and security monitoring.
- These different security controls help to manage the circumventing of security protocols.

❖ **System hardening and monitoring**

- The underlying architecture provides additional access to the DBMS.
- It is vital that all systems are patched consistently, hardened using known security configuration standards, and monitored for access, including insider threats.

❖ **DBMS configuration**

- It is critical that the DBMS be properly configured and hardened to take advantage of security features and limit privileged access that may cause a mis -configuration of expected security settings.
- Monitoring the DBMS configuration and ensuring proper change control processes helps ensure that the configuration stays consistent.

❖ **Authentication**

- Database security measures include authentication, the process of verifying if a user's credentials match those stored in your database, and permitting only authenticated users access to your data, networks, and database platform.

❖ **Access**

- A primary outcome of database security is the effective limitation of access to your data.

- Access controls authenticate legitimate users and applications, limiting what they can access in your database.
- Access includes designing and granting appropriate user attributes and roles and limiting administrative privileges.

❖ **Database auditing**

- Monitoring (or auditing) actions as part of a database security protocol delivers centralized oversight of your database.
- Auditing helps to detect, deter, and reduce the overall impact of unauthorized access to your DBMS.

❖ **Backups**

- A data backup, as part of your database security protocol, makes a copy of your data and stores it on a separate system.
- This backup allows you to recover lost data that may result from hardware failures, data corruption, theft, hacking, or natural disasters.

❖ **Encryption**

- Database security can include the secure management of encryption keys, protection of the encryption system, management of a secure, off-site encryption backup, and access restriction protocols.

❖ **Application security**

- Database and application security framework measures can help protect against common known attacker exploits that can circumvent access controls, including SQL injection.

Why is database security important?

- Database security can guard against a compromise of your database, which can lead to financial loss, reputation damage, consumer confidence disintegration, brand erosion, and non-compliance of government and industry regulation.
- Database security safeguards defend against a myriad of security threats and can help protect your enterprise from:
    - ✓ Deployment failure

- ✓ Excessive privileges
- ✓ Privilege abuse
- ✓ Platform vulnerabilities
- ✓ Unmanaged sensitive data
- ✓ Backup data exposure
- ✓ Weak authentication
- ✓ Database injection attacks

Security Levels

To protect the database, we must take security measures at several levels:

1. **Physical:**

   The sites containing the computer systems must be secured against armed or surreptitious entry by intruders.

2. **Human:**

   Users must be authorized carefully to reduce the chance of any such user giving access to an intruder in exchange for a bribe or other favors .

3. **Operating System**:

   No matter how secure the database system is, weakness in operating system security may serve as a means of unauthorized access to the database.

4. **Network:**

   Since almost all database systems allow remote access through terminals or networks, software-level security within the network software is as important as physical security, both on the Internet and in networks private to an enterprise.

5. **Database System:**

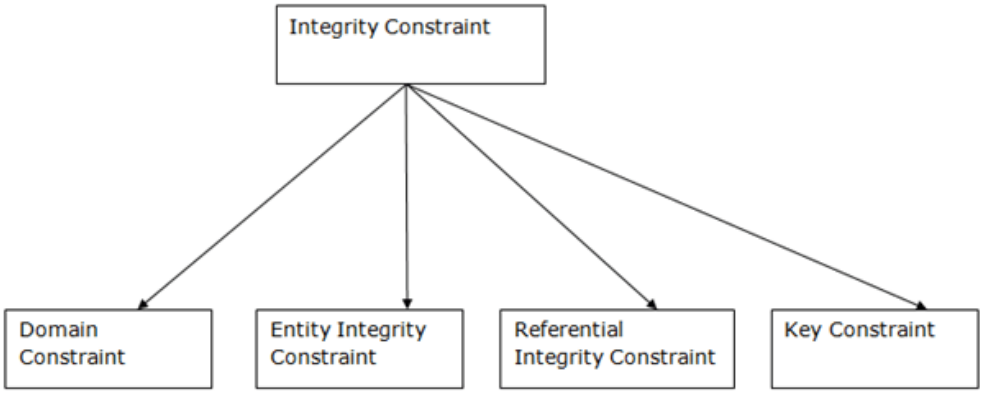
   Some database-system users may be authorized to access only a limited portion of the database. Other users may be allowed to issue queries, but may be forbidden to modify the data. It is responsibility of the database system to ensure that these authorization restrictions are not violated.

Security at all these levels must be maintained if database security is to be ensured. A weakness at a low level of security (physical or human) allows circumvention of strict high level (database) security measures.

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, time, 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

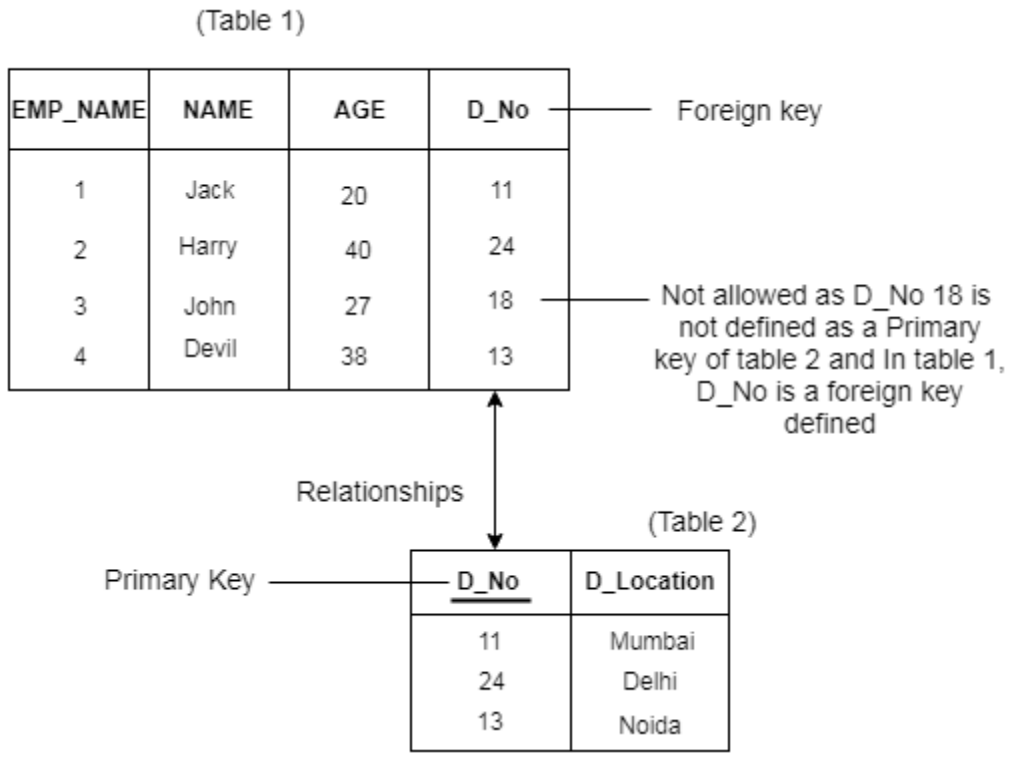
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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:



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

Comparison Between Security and Integrity

- The key **difference between Security and Integrity** is that **Data security** is the protection of data from **unauthorized users**.
- **Data integrity** means that the data contained in the database is both correct and consistent.
- For this purpose, the data stored in the database must satisfy certain types of constraints (rules).

Security	Integrity
Data security defines the prevention of data corruption through the use of controlled access mechanisms.	Data integrity defines the quality of data, which guarantees the data is complete and has a whole structure.
Data security deals with the protection of data	Data integrity deals with the validity of data
Data security is making sure only the people who should have access to the data are the only ones who can access the data.	Data integrity is making sure the data is correct and not corrupt.
Data security refers to making sure that data is accessed by its intended users, thus ensuring the privacy and protection of data.	Data integrity refers to the structure of the data and how it matches the schema of the database.
Authentication / authorization, encryptions, and masking are some of the popular means of data security.	Backing up, designing a suitable user interface and error detection/correction in data are some of the means to preserve integrity.

Encryption

- The various provisions a database system may make for authorization may not be sufficient protection or highly sensitive data.
- In such cases, data may be encrypted. It is not possible for encrypted data to be read unless the reader knows how to decipher (decrypt) the encrypted data.
- There are a vast number of techniques for the encryption of data.
- Simple techniques for encryption may not provide adequate security since it may be easy for an authorized user to break the code.

- As an example of a bad encryption technique, consider substitution of each character with the next character in the alphabet. Thus:

  Perryridge

  becomes:

  Qfsszshehf

- If an unauthorized user sees only “Qfsszsjehf,” there is probably insufficient information to break the code.
- However, if the intruder sees a large number of encrypted branch names, the intruder could use statistical data regarding the relative frequency of characters (for example, “e” is more common than “x”) to guess what substitution is being made.
- Good encryption techniques have the following properties:

  - ✓ It is relatively simple for authorized users to encrypt and decrypt data.
  - ✓ The encryption scheme depends not on the secrecy of the algorithm but on a parameter of the algorithm called the encryption key.
  - ✓ It is extremely difficult for an intruder to determine the encryption key.

- The Data Encryption Standard is an approach which does both a substitution of characters and a rearrangement of their order based on an encryption key.
- In order for this scheme to work, the authorized users must be provided with the encryption key via a secure mechanism.
- This is a major weakness since the scheme is no more secure than the secureness of the mechanism by which the encryption key is transmitted.
- There is an alternative scheme that avoids some of the problems with the Data Encryption Standard.
- This scheme, called public–key encryption, is based on two keys, a public key and private key.
- Each user has his or her own public key and private key .
- All public keys are published. Each user’s private key is known only to the one user to whom the key belongs.

- If user 1 wants to store encrypted data, encrypts it using his or her public key 1. decryption requires the private key 1.
- Because the encryption key for each user is public, it is possible to exchange information securely using this scheme.
- if user 1 wants to share data with 2 , U1 encrypts the data using , the public key of 2.
- Since only user knows how to decrypt the data, secure information transfer is accomplished.
- For public key encryption to work, there must be a scheme for encryption that can be made public without making it easy to figure out the scheme for decryption.
- Such a scheme does exist. It is based on the following:
    - ✓ There is an efficient algorithm for testing whether or not a number is prime
    - ✓ No efficient algorithm is known for finding the prime factors of a number.
- Data is treated as a collection of integers for purposes of this scheme. A public key is created by computing the product of two large prime numbers 1 and 2 .
- The private key consists of the (1,2) , and the decryption algorithm cannot be used successfully if only the product is known 12.
- Since all that is published in the product , an unauthorized user would need to be able to factor 12 in order to steal data. By choosing and to be sufficiently large (over 100 digits).