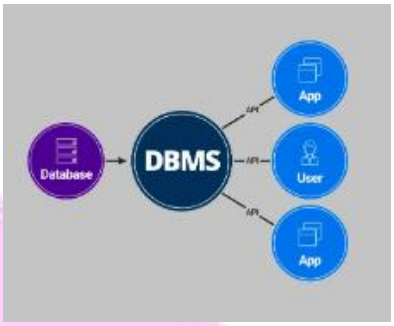
Introduction to Database

Data:- Data is unorganized, raw or isolated facts about anything.
For example:- text, audio, video, images, student name, marks etc.



Information:- Information is the processed data that has been converted into more useful or intelligent form.
For example: **Report card sheet.**

Database:- A database is an organized collection of data, so that it can be easily accessed and managed and updated.

Database management system:- Database management system is a software which is used to manage the database. For example: MySQL, Oracle, SQL server, DB2 etc
DBMS is used to store, retrieve, and manage data efficiently and effectively.

Traditional File-Based System vs. DBMS

Feature	File-Based System	DBMS
Redundancy	High redundancy	Redundancy minimized
Consistency	Data inconsistencies common	Maintains data consistency
Security	Limited security features	Advanced security mechanisms
Scalability	Poor scalability	Easily scalable
Performance	Slow for complex queries	Optimized query performance

Components of DBMS

1. **Database Engine:**
   - Handles data storage, retrieval, and modification.
   - Executes database queries efficiently.
2. **Database Schema:**
   - Defines the logical structure of the database (tables, relationships, constraints).
3. **Query Processor:**
   - Interprets and executes database queries written in SQL.
   - Converts high-level queries into machine-level instructions.
4. **Data Dictionary:**
   - Stores metadata (e.g., table names, data types, constraints).
   - Helps maintain data consistency.
5. **Transaction Manager:**
   - Ensures ACID properties (Atomicity, Consistency, Isolation, Durability) for transactions.
6. **Storage Manager:**
   - Manages physical storage of data and maintains efficient data retrieval mechanisms.

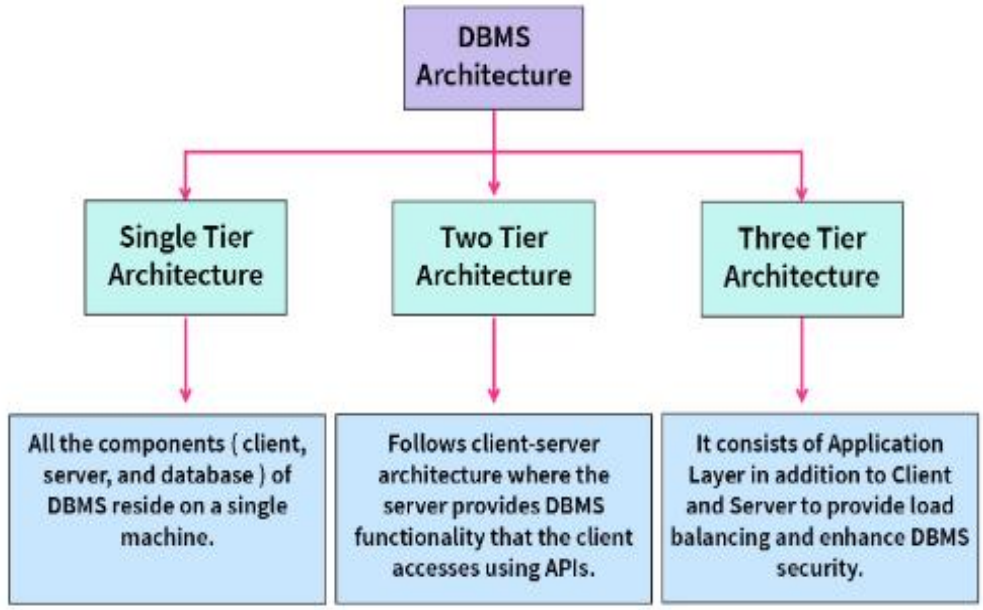
Advantages of DBMS

1. Improved Data Management
2. Data Integrity and Accuracy
3. Data Security
4. Minimized Data Redundancy
5. Supports Concurrent Access
6. Scalability and Flexibility

Drawbacks of DBMS

1. Complexity
2. Cost
3. Performance
4. Dependence on Vendors

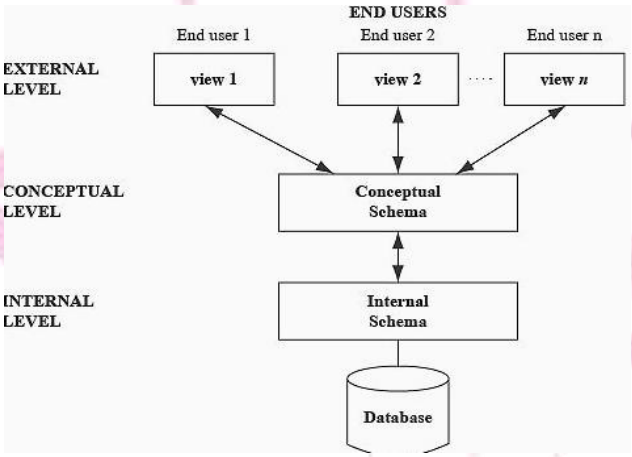
DBMS Architecture



The 3-Level Architecture of DBMS

- Also known as the ANSI/SPARC architecture.
- It provides a framework to separate the user's view, logical structure, and physical storage of data.
- This separation ensures data independence, efficient management, and flexibility.

1. **external level (Presentation or user or view level)**
2. **conceptual level (Application level or logical level)**
3. **Physical level (Database level or internal level)**



1. External Level (User View)

- The external level defines how individual users or user groups view the data.
- Each user can have a customized view based on their needs.
- Multiple views can exist for the same database.
- Provides security by restricting access to only the data required by users.

Example1:

A student may view their course details, while an admin views all student records.

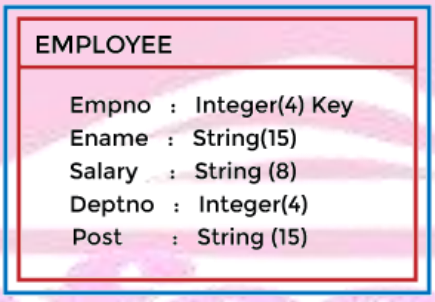
Example2:

2. Conceptual Level (Logical Level)

- The conceptual level represents the entire database's logical structure. It describes what data is stored, the relationships between data, and the constraints applied.
- It focuses on entities, attributes, relationships, and constraints.
- It is independent of how data is physically stored.
- All external views map to the conceptual level.
- Ensures logical data independence.

Example: Logical representation of a student entity:
Entity: Student.
Attributes: Student_ID, Name, Age, Course.
Relationship: Enrolled in Course.

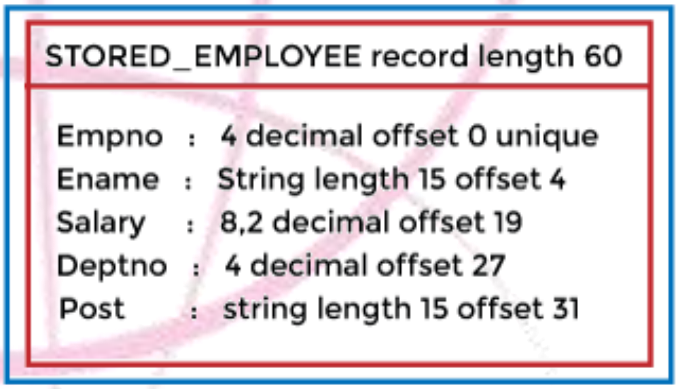
Example2:



3. Internal Level (Physical Level)

- The internal level describes how data is physically stored in the database. It deals with file structures, indexes, storage allocation, and optimization techniques.
- It focuses on data storage and retrieval mechanisms.
- It manages storage devices (e.g., disks, memory).
- It provides physical data independence.

Example:
Data stored as files with specific block sizes and index structures for fast retrieval.
B-trees or hash indexing may be used.



Data Independence in 3-Level Architecture

1. **Logical Data Independence:** The ability to change the conceptual schema without affecting the external schemas (user views).
   Example: Adding a new column to a table without affecting existing views.

2. **Physical Data Independence:** The ability to change the internal schema (storage structure) without affecting the conceptual schema.
   Example: Moving from magnetic disks to SSDs without impacting database queries.

Mapping Between the Levels

1. External-Conceptual Mapping:
- Maps each external view to the conceptual schema.
- It ensures that changes in user views are reflected in the conceptual schema.

2. Conceptual-Internal Mapping:
- Maps the conceptual schema to the internal schema.
- It ensures that logical data structures match physical storage structures.

Advantages of 3-Level Architecture

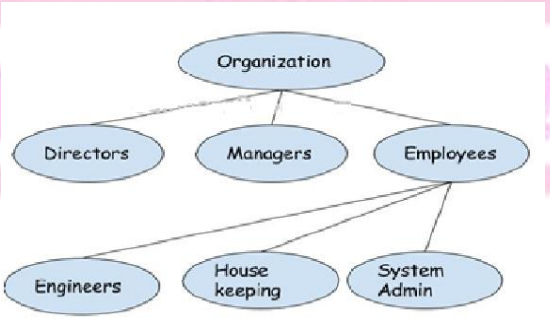
1. **Data Independence:** Allows changes at one level without affecting other levels.
2. **Improved Security:** External level provides restricted access based on user roles.
3. **Simplified Database Design:** Conceptual schema offers a unified logical view.
4. **Flexibility:** Different user views can be created easily.
5. **Better Performance Optimization:** Internal level manages data storage efficiently.

Types of DBMS

1. Hierarchical DBMS

- Data is organized in a tree-like structure, with parent-child (one to many) relationships.
- Each child record has only one parent.
- Relationships are represented by links between records.
- Data access follows a top-down approach.
- **Example:** IBM Information Management System (IMS).

Organization hierarchy: Root: CEO, Level 1: Managers, Level 2: Employees



Advantages:
- Efficient for one-to-many relationships.
- Fast access to data through predefined paths.

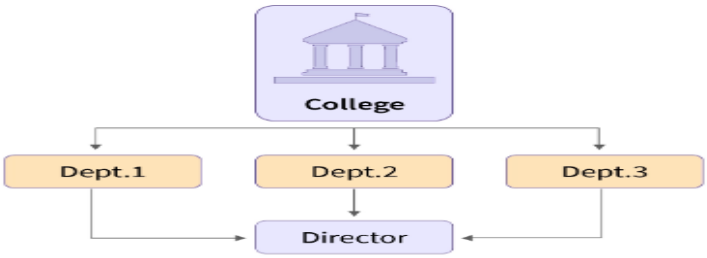
Disadvantages:
- Rigid structure; difficult to modify.
- Requires navigating the entire hierarchy for queries.

2. Network DBMS

Data is represented using a graph structure, allowing multiple parent-child relationships (many-to-many relationships).
Uses pointers to connect records.
More flexible than hierarchical DBMS.
Example: Integrated Data Store (IDS).

Advantages:
Supports complex relationships.
More flexible navigation compared to hierarchical DBMS.

Disadvantages:
Complex implementation.
Difficult to modify pointers during updates.



3. Relational DBMS (RDBMS)

- It stores data in tables (relations) with rows and columns. Relationships are established using keys.
- Data is accessed using Structured Query Language (SQL).
- Data is stored in tables with unique rows identified by a primary key.
- Relationships are established using foreign keys.
- **Example:** MySQL, Oracle, Microsoft SQL Server.

Student table

ID	Name	Age	Address
100	saroj	29	Lucknow
101	Jeeva	30	Kanpur
102	prem	28	Agra
103	prabhu	30	Noida

Advantages:

- Simple and intuitive table-based structure.
- Supports complex queries using SQL.
- Ensures data integrity and normalization.

Disadvantages:

- Performance issues with very large datasets.
- Requires more computational resources.

4. Object-Oriented DBMS (OODBMS)

- Stores data as objects, similar to object-oriented programming. Objects can include attributes and methods.
- Supports concepts like inheritance, encapsulation, and polymorphism.
- Data and behavior (methods) are stored together.
- Example: ObjectDB, db4o.

Advantages:

- Suitable for complex data types like multimedia and graphics.
- Reusability through object inheritance.

Disadvantages:

- Complex to implement and manage.
- Not as widely adopted as RDBMS.

5. NoSQL DBMS

- It refers to databases that do not use traditional relational models. It includes document-based, key-value, column-oriented, and graph databases.
- Supports unstructured and semi-structured data.
- Highly scalable and flexible.
- Example: MongoDB, Cassandra, HBase, Redis, Amazon DynamoDB, Neo4j, Amazon Neptune.

Advantages:

- Flexible schema.
- High scalability and performance for distributed systems.

Disadvantages:

- Less support for ACID transactions.
- Complex querying compared to SQL.

Data Model

A Data Model defines how data is logically structured, organized, and manipulated in a database. It acts as a blueprint for designing databases and provides rules for storing and retrieving data.

Some of the Data Models in DBMS are:

1. Hierarchical Model
2. Network Model
3. Entity-Relationship Model
4. Relational Model
5. Object-Oriented Data Model

Entity Relationship model

- **The Entity-Relationship (ER) Model is a high-level data model used to represent the logical structure of a database.**
- **It visually describes data, relationships, and constraints using diagrams.**
- **The ER Model simplifies the design process by focusing on entities, attributes, and relationships.**

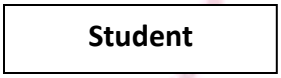
Components of the ER Diagram

This model is based on three basic concepts:

1. Entities
2. Attributes
3. Relationships

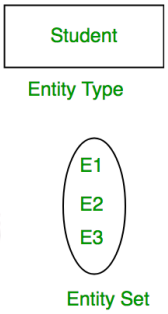
1. Entity

- An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
- Entity is a thing in the real world with an independence existence.
- The collection of entities that have the same attributes is called an entity set.
- Examples of entities:
  - ✓ Person: Employee, Student, Patient
  - ✓ Place: Store, Building
  - ✓ Object: Machine, product, and Car
  - ✓ Concept: Account, Course



Entity set

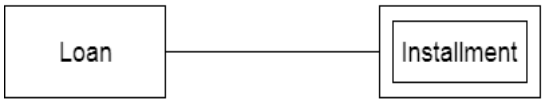
- Entity set is a collection of entities of a particular entity type at a point in time.
- An Entity is an object of Entity Type and set of all entities is called as entity set. e.g. E1 is an entity having Entity Type Student and set of all students is called Entity Set.
- For example:- a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties.



Types of Entities

Strong Entities:

- Exist independently in the database.
- It is represented by a rectangle symbol.
- Have a primary key to uniquely identify each instance.
- The relationship between two strong entity set shown by using a diamond symbol.
- The connecting line of the strong entity set with the relationship is single.
- The member of a strong entity set is called as dominant entity set.
- Example: Student, Employee.



Weak Entities:

- Depend on a strong entity for existence.
- It does not have enough attributes to build a primary key.
- Identified by a foreign key and a discriminator attribute.

- It is represented by a double rectangle symbol.
- The relationship between one strong and a weak entity set shown by using the double diamond symbol.
- The line connecting the weak entity set for identifying relationship is double.
- The member of a weak entity set called as a subordinate entity set.
- Example: Dependent (in relation to Employee).

2. Attributes

- Attributes are the properties which define the entity.
- An entity is represented by a set of attributes.
- An attribute in ER Diagram is represented by an Ellipse
- For example: A student entity has attributes like Roll_No, Name, DOB, Age, Address, Mobile_No etc.
- For example: a lecture might have attributes: time, date, duration, place, etc.
- 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.
- An entity may contain any number of attributes.
- The attributes that can uniquely define an entity are considered as the primary key.

Types of attributes

There are some following types of attributes:

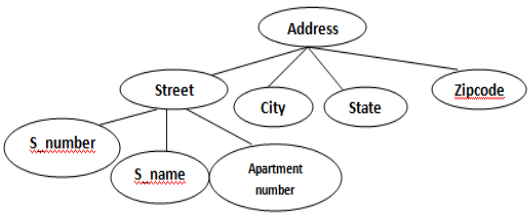
1. Simple attribute
2. Composite attribute
3. Derived attribute
4. Stored Attributes
5. Single-value attribute
6. Multi-value attribute
7. Key Attribute

1. Simple attribute

- Simple attributes are atomic values, which cannot be divided further.
- It is also called a key attribute. It modelled in ER diagram as a simple eclipse with underlined attribute name.
- **For example:-** a student's phone number is an atomic value of 10 digits. The roll number of a student, the id number of an employee, weight of a person etc.

2. Composite attribute

- Composite attributes are made of more than one simple attribute.
- It can be divided into sub-attributes which represent more basic attributes with independent meanings.
- In other words an attribute that is composed of several other attributes is known as a composite attribute.
- The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.
- **Example:** address(city, street pincode), name(Fname, Mname, Lname)



3. 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.
- The value of a derived attribute is not stored, but is computed when required.
- It is represented with dashed ovals in an ER Diagram.
- **For example:-** average_salary in a department should not be saved directly in the database, instead it can be derived. Average marks of a student is derived from all subject marks of a student entity and An age can be derived from data_of_birth(DOB).

4. Base or Stored Attributes

- Base or stored attributes are the attributes for which the value of other attributes is derived.
- **For example:-** Date of birth of a person is base attribute as the age is derived from DOB.

5. Single-value attribute

- Single-value attributes contain single value.
- Attributes having single value for a particular entity instance is known as single-valued attribute.
- **For Example:** the age of a person, Social_Security_Number,aadhar card number, weight of person.

6. Multi-value attribute

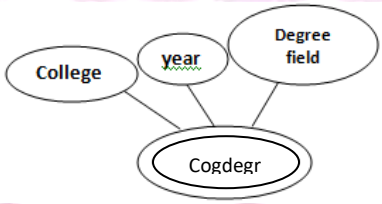
- Multi-value attributes may contain more than one value.
- In ER diagram, multivalued attribute is represented by double oval.
- **For example:-** a person can have more than one phone number, email_address, college degree, languages etc.

7. Key Attribute

- Key attribute uniquely identifies an entity from an entity set.
- A key attribute can uniquely identify an entity from an entity set.
- The key attribute is represented by an ellipse with the text underlined.
- For example:- student roll number can uniquely identify a student from a set of students.

8. Complex Attributes

- Multivalued and composite components are called complex components.
- Multivalued attributes: represented with { }
- Composite attributes: represented with ( )
- For Example:- {Collegedegree(college, year, degree field)}.

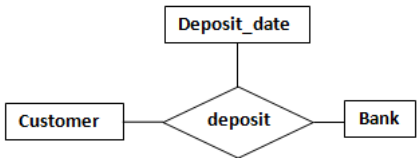


9. Null values

- Null is something which is not applicable or unknown.
- An attribute takes a null value when an entity does not have a value for it. The null value may indicate “not applicable” i.e the value does not exist for the entity.
- For example:- one may have no middle name.
- Null can also designate that an attribute value is unknown. An unknown value may be either missing (the value does exist, but we do not have that information) or not known (we do not know whether or not the value actually exists).
- For example:- if the name value for a particular customer is null, we assume that the value is missing, since every customer must have a name.
-

10. Descriptive attribute

- A property or characteristic of a relationship (versus of an entity) is defined by descriptive attribute.
- **For Example:-** A customer deposits some money to the bank customer and bank are two entity and deposit is a relationship. Deposit date is a descriptive attribute for relation deposit because when we deposit money then only we get need deposit date.

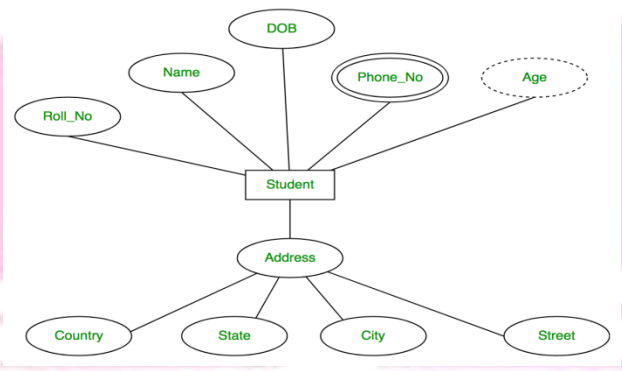


Shapes used in ER Diagram

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set

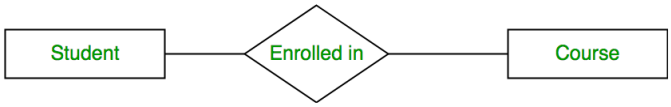
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multivalued Attributes
Dashed Ellipses: Derived Attributes
Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

All type of attributes in one diagram:-



3. Relationship

- A relationship is an association among several entities. The diamond shape showcases a relationship in the ER diagram.
- **For example:** an employee works_at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.
- A relationship type represents the association between entity types.

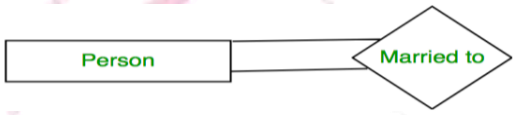


Degree of Relationship

- The number of participating entities in a relationship defines the degree of the relationship.
- We can divide degree of relationship in the following type:

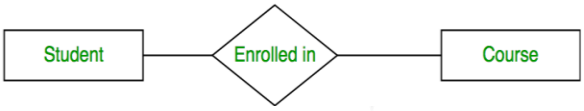
1. Unary Relationship

- 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.



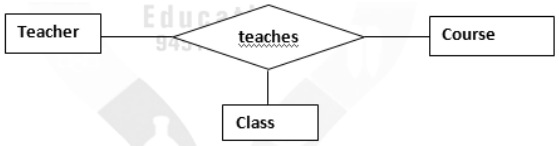
2. Binary Relationship

- 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. Ternary relationship

- When there are three entities set participating in a relation, the relationship is called as ternary relationship.
- **For example:-** Teacher teaches course and also a teacher teaches to a class.



Constraints

- Constraints are the set of rules that ensures data consistency.
- An E-R enterprise schema may define certain constraints to which the contents of a database must conform. Here we have two of the most important types of constraints for relationship.

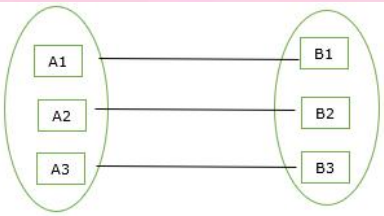
a) Mapping cardinalities and
b) Participation constraints

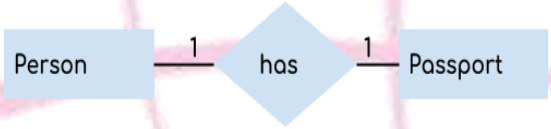
a) Mapping Cardinality

- The number of times an entity of an entity set participates in a relationship set is known as cardinality.
- Mapping cardinality is most useful in describing binary relation sets, although they can contribute to the description of relation sets containing more than two entity sets.
- Cardinality can be of following 4 types:

1. One-to-One Relationship

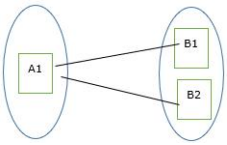
- When a single element of an entity is associated with a single element of another entity, it is called a one-to-one relationship.
- **For example:-** a student has only one identification card and an identification card is given to one person. A male can marry to one female and a female can marry to one male.





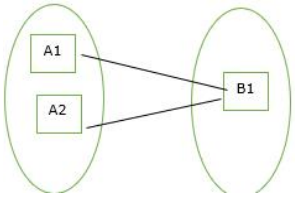
2. One to Many Relationship

- When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship.
- **For example:-** a customer can place many orders but a order cannot be placed by many customers.



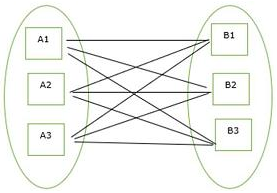
3. Many to One Relationship

- When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship.
- **For example:-** many students can study in a single college but a student cannot study in many colleges at the same time.



4. Many to Many Relationship

- When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship.
- **For example:-** A can be assigned to many projects and a project can be assigned to many students.



Minimum Cardinality

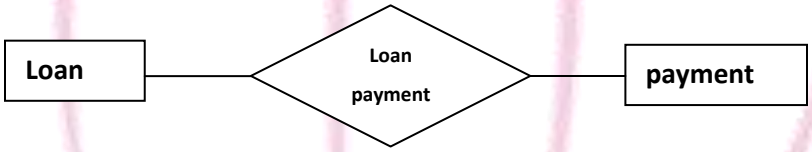
- Minimum cardinality is a constraint that defines the minimum number of identifiers or instances participates in a relationship. Minimum cardinality can be zero also.
- For example:- Every Person need to have an Address, therefore its minimum cardinality is 1.

Maximum Cardinality

- Maximum cardinality is a constraint that defines the maximum number of identifiers or instances participates in a relationship.
- For example:- A Person can have more than one house, therefore its maximum cardinality is maximum number of house.

Existence Dependencies

- If the existence of entity x depends on the existence of entity y, then x is said to be existence dependent on y.
- ✓ y is a dominant entity (in example below, loan)
- ✓ x is a subordinate entity (in example below, payment)
- If a loan entity is deleted, then all its associated payment entities must be deleted also.



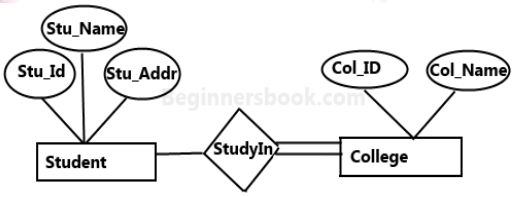
b) Participation constraints

Participation of an entity set represents that how an entity in entity set is participating in a relationship set.
There are 2 types of participation constraints.

a) Total Participation
b) Partial Participation

a) Total Participation of an Entity set

- Each entity in entity set must have at least one relationship in a relationship set.
- It is also called mandatory participation.
- **For example:-** In the following diagram each college must have at-least one associated Student. Total participation is represented using a double line between the entity set and relationship set.



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

b) Partial participation of an Entity Set

- Partial participation of an entity set represents that each entity in the entity set may or may not participate in the relationship instance in that relationship set.
- It is also called as optional participation.
- Partial participation is represented using a single line between the entity set and relationship set.

For Example:- Consider an example of an IT company. There are many employees working for the company. Let's take the example of relationship between employee and role software engineer. Every software engineer is an employee but not every employee is software engineer as there are employees for other roles as well, such as housekeeping, managers, CEO etc. So we can say that participation of employee entity set to the software engineer relationship is partial.

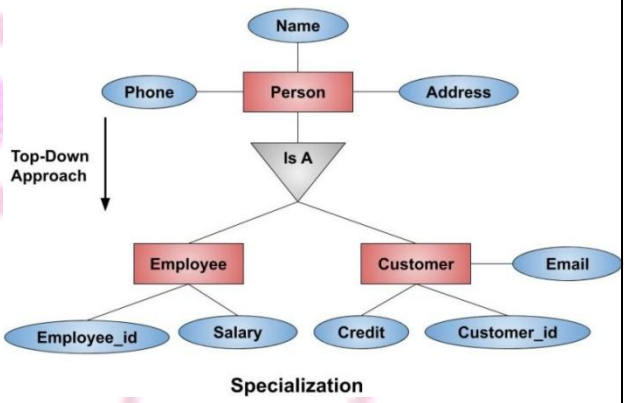
Extended or Enhanced ER Model

- As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling.
- Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.

- Three new concepts were added to the existing ER Model, they were:
    1. **Specialization**
    2. **Generalization**
    3. **Aggregation**

1. Specialization

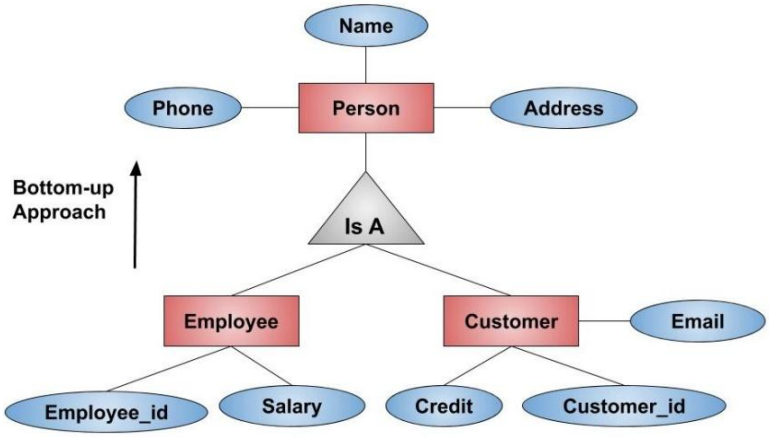
- Breaks a higher-level entity into lower-level entities based on special features.
- Top-down: A single entity is divided into subclasses, inheriting attributes of the superclass.
- Helps share common attributes among entities in the database.
- Identifies subsets of entities with distinguishing characteristics.
- Superclass → Subclass (using "is a" relationship).
- Attribute Inheritance: Subclasses inherit all attributes and relationships of the superclass.
- Represented by a downward triangle symbol.
- Refines schema design by repeatedly applying specialization.



Specialization

2. Generalization

- Combines two or more lower-level entities to form a higher-level entity.
- Bottom-up: Subclasses merge into a generalized superclass.
- Opposite of specialization.
- A higher-level entity can combine with lower-level entities to form a new superclass.
- Example: Faculty and Student → generalized to Person.
- Common features like Name, Phone, and Address are shared at a higher level.

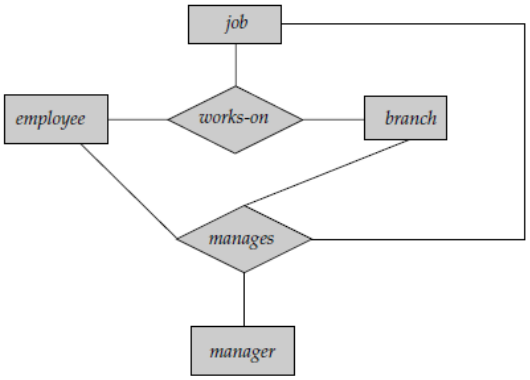


Generalization

3. Aggregation

- One limitation of the E-R model is that it cannot express relationships among relationships.
- Aggregation allows representing relationships among relationships by treating a relationship set as a higher-level entity.
- In aggregation, the relation between two or more entities is treated as a single entity for abstraction.

For example:- consider the ternary relationship works-on, between an employee, branch, and job. Now, suppose we want to record managers for tasks performed by an employee at a branch; that is, we want to record managers for (employee, branch, job) combinations.



Benefits of Aggregation

- Simplifies complex relationships.
- Reduces redundancy by grouping related information.
- Enables efficient representation of real-world scenarios involving relationships of relationships.

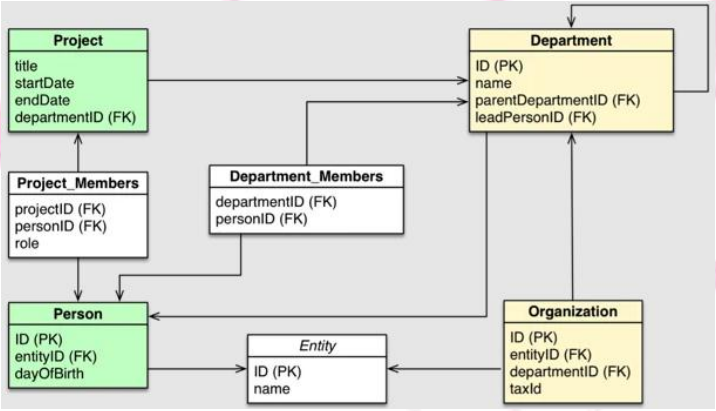
Relational Model

Database Schema: A database schema describes the structure of the database, including tables, attributes, and relationships.

Types of Database Schema

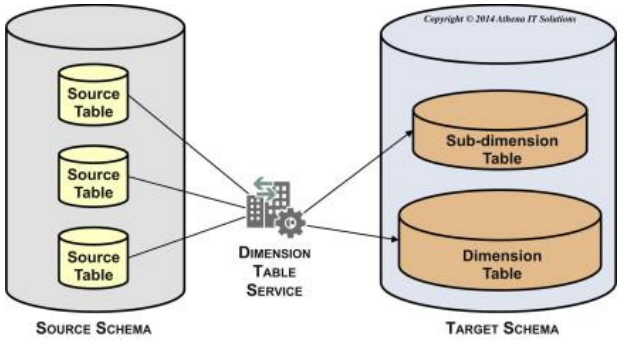
Logical Schema:

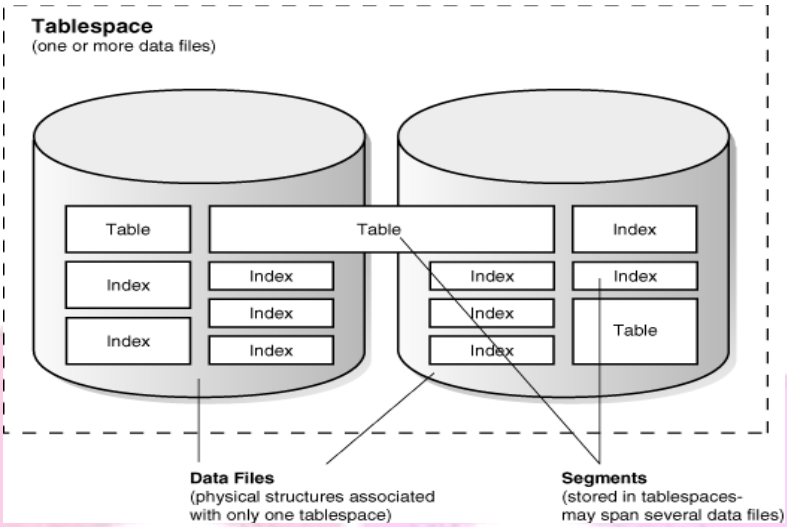
- Defines the logical structure of the database.
- Focuses on entities, attributes, and relationships.
- **Example:** A schema defining employee, project, organization and department tables with their attributes.



Physical Schema:

- Defines how the data is physically stored in the database.
- Includes details like file structures, indexes, and storage paths.
- **Example:** Storage of student data in specific data blocks on disk.





Concept of Relational Model

1. **Table (Relation):**
   - A collection of data organized in rows and columns.
   - Represents a single entity or concept.
   - Each table has a unique name.

StudentID	Name	Age	Course
101	Alice	21	CS
102	Bob	22	IT

2. **Row (Tuple):**
   - A single record or instance in a table.
   - Represents a specific data entry for the entity.

3. **Column (Attribute):**
   - A field or property of the entity.
   - Each column has a unique name and a specific data type (e.g., integer, string).

4. **Domain:**
   - The set of all possible values an attribute can take.
   - Example: The domain of the Age attribute could be integers from 0 to 150.

5. **Schema:**
   - A logical description of the structure of a database, including tables, attributes, and relationships.

6. **Keys:**
   - **Primary Key:** Uniquely identifies each row in a table.
   - **Foreign Key:** Establishes a relationship between two tables.
   - Candidate Key, Composite Key, Alternate Key, and Surrogate Key

7. **Relational Operations:**
   - **Selection (σ):** Extracts rows that satisfy a condition.
   - **Projection (π):** Extracts specific columns from a table.
   - **Join (⋈):** Combines rows from two tables based on a related attribute.
   - **Union, Intersection, Difference:** Set-based operations on relations.

Properties of Relations

- **Uniqueness:** Each row (tuple) in a relation is unique.
- **Atomicity:** Each column value is atomic (indivisible).
- **No Duplicate Columns:** Each column (attribute) has a unique name.
- **Order Irrelevance:** The order of rows and columns does not matter in a relation.
- **Null Values:** A column can have a null value, indicating missing or unknown data.

Types of Keys

Example: Student Table

SID	Name	Email	Phone	CourseID	Aadhar_no
101	Alice	alice@gmail.com	9876543210	CSE01	23456
102	Bob	bob@gmail.com	9876543211	CSE02	23631
103	Charlie	charlie@gmail.com	9876543212	CSE01	98766
104	David	david@gmail.com	9876543213	CSE03	56433

1. **Super Key:**

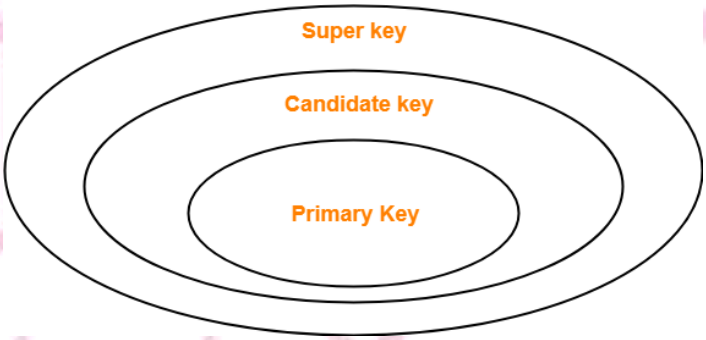
- A set of one or more attributes that uniquely identifies a tuple.
- **Example:** {SID}, {Aadhar_no}, {SID, Name}, {SID, Aadhar_no, Name}, {SID, Name, email},{SID, Aadhar_no, Name, phone}, {SID, Name, email, phone}, {Name, email, phone}, {Name, email} etc.

2. **Candidate Key:**

- A minimal super key (no extra attribute).
- Example: {SID}, {Aadhar_no}, {email} etc.

3. **Primary Key:**

- A candidate key chosen to uniquely identify tuples.
- Cannot have null values.
- Example: SID in the student table.



Example: Student Table(referencing table)

SID	Name	Email	Phone	CourseID	Aadhar_no
101	Alice	alice@gmail.com	9876543210	CSE01	23456
102	Bob	bob@gmail.com	9876543211	CSE02	23631
103	Charlie	charlie@gmail.com	9876543212	CSE01	98766
104	David	david@gmail.com	9876543213	CSE03	56433

Example: Course Table(referenced table)

CourseID	CourseName	Department	Credits
CSE01	Computer Science	CSE	4
CSE02	Information Tech	IT	3
CSE03	Electronics	ECE	4
CSE04	Mechanical Engg	ME	3

4. **Foreign Key:**
- An attribute in one table that refers to the primary key in another table.
- Foreign key can take the NULL value.
- Foreign key may have a name other than that of a primary key.
- There is no restriction on a foreign key to be unique.
- Referenced relation may also be called as the master table or primary table.
- Referencing relation may also be called as the foreign table.
- Values in the Foreign Key must match values in the Primary Key of the referenced table.
- Ensures referential integrity.
- **Example:** CourseID in a student table referencing the primary key in the course table.

5. **Composite Key:**
- A primary key that consists of two or more attributes.
- Whenever a primary key consists of more than one attribute, it is known as a composite key.
- This key is also known as Concatenated key.
- **Example:** {name, phone} in an order details table.

6. **Alternate Key:**
- Alternate keys are nothing but candidate keys that are not chosen as primary key.
- It is also called secondary key.
- Example: If Email is a candidate key but not the primary key.

7. **Unique Key:**
- Similar to the primary key but allows one null value.
- It is unique for all the records of the table.
- Once assigned, its value cannot be changed i.e. it is non-updatable.
- It may have a NULL value.
- Example: Email, phone

8. **Surrogate Key:**
- A system-generated key, usually a unique number.
- A surrogate key is an artificial or synthetic key used to uniquely identify records in a table.
- Automatically generated by the database upon record insertion.
- Typically an integer, with no inherent meaning to the data.
- Always unique, non-null, and updatable.
- When no suitable natural primary key exists automatically generated.
- When the natural key is too large or complex to use efficiently.
- Example: Auto-incremented ID.

Properties of Relations:
- **Uniqueness:** Each row (tuple) in a relation is unique.
- **Atomicity:** Each column value is atomic (indivisible).
- **No Duplicate Columns:** Each column (attribute) has a unique name.
- **Order Irrelevance:** The order of rows and columns does not matter in a relation.
- **Null Values:** A column can have a null value, indicating missing or unknown data.

Relational Integrity Constraints: Rules that ensure the accuracy and consistency of data in a relational database.

(a) Domain Integrity:

- Ensures that the values in a column fall within a specific range or domain.
- **Example:** A DateOfBirth column must contain valid dates.

(b) Referential Integrity:

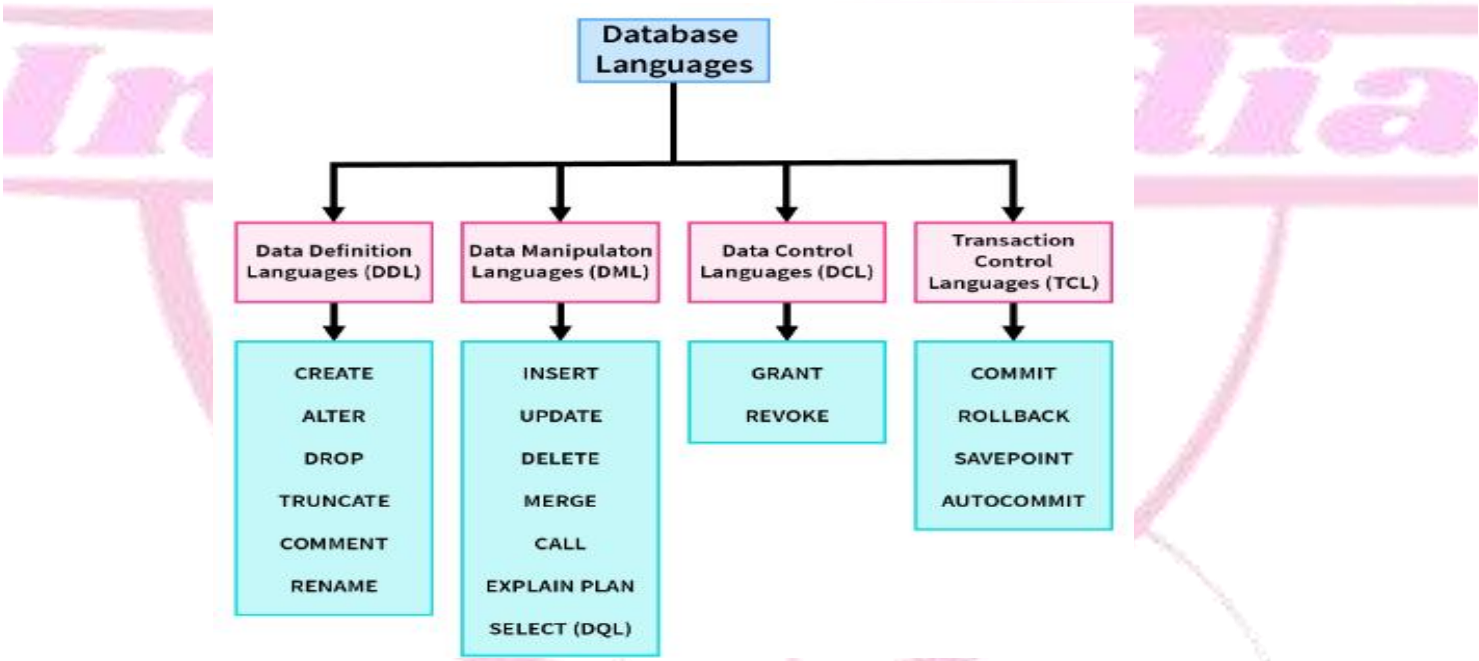
- Ensures that a foreign key value always refers to an existing primary key in another table.
- **Example:** A StudentID in the enrollment table must match an existing StudentID in the student table.

(c) Entity Integrity:

- Ensures that the primary key is unique and not null for every tuple.
- **Example:** Each EmployeeID in an employee table must be unique and not null.

Database Language

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).



There are 4 types of database languages:-

1. **Data Definition Language (DDL)**
2. **Data Manipulation Language (DML)**
3. **Data Control Language (DCL)**
4. **Transaction Control Language (TCL)**

1. Data Definition Language (DDL)

- Data Definition Language (DDL) is a set of special commands that allows us to define and modify the structure and the metadata of the database.
- These commands can be used to create, modify, and delete the database structures such as schema, tables, indexes, etc.
- Every change implemented by a DDL command is auto-committed (the change is saved permanently in the database), these commands are normally not used by an end-user.
- **Some of the DDL commands are:**

a) CREATE:-

- It is used to create the database or its schema objects.
- ***To create a new database:***

```
CREATE DATABASE database_name;
```

- ***To create a new table:***

```
CREATE TABLE table_name (
  column_1 DATATYPE,
  column_2 DATATYPE,
  column_n DATATYPE );
```

b) DROP:-

- It is used to delete the database or its schema objects.
- ***To delete an object:***

```
DROP object object_name
```

- ***To delete an existing table:***

```
DROP TABLE table_name;
```

- ***To delete the whole database:***

```
DROP DATABASE database_name;
```

c) ALTER:-

- It is used to modify the structure of the database objects.
- ***To add new column(s) in a table:***

```
ALTER TABLE table_name ADD (
  column_1 DATATYPE,
  column_2 DATATYPE,
  column_n DATATYPE );
```

- ***To change the datatype of a column in a table:***

```
ALTER TABLE table_name
MODIFY column_name DATATYPE;
```

- ***To remove a column from a table:***

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

d) TRUNCATE:-

- It is used to remove the whole content of the table along with the deallocation of the space occupied by the data, without affecting the table's structure.
- ***To remove data present inside a table:***

```
TRUNCATE TABLE  table_name;
```

NOTE:- We can also use the DROP command to delete the complete table, but the DROP command will remove the structure along with the contents of the table. Hence, if you want to remove the data present inside a table but not the table itself, you can use TRUNCATE instead of DROP.

e) COMMENT:-

- It is used to add comments about the tables, views, and columns into the data dictionary. These comments can help the developers to better understand the structure of the database.

- ***To comment on table/view:***
  COMMENT ON TABLE table_name IS 'text';
- ***To comment on a column:***
  COMMENT ON COLUMN table_name.column_name IS 'text';
- ***To drop a comment:***
  COMMENT ON TABLE table_name IS ' ';

f) RENAME:-

- It is used to change the name of an existing table or a database object.
- ***To rename a table:***
  RENAME old_table_name TO new_table_name;
- ***To rename a column of a table:***
  ALTER TABLE table_name
  RENAME COLUMN old_Column_name to new_Column_name;

2. Data Manipulation Language (DML)

- Data Manipulation Language (DML) is a set of special commands that allows us to access and manipulate data stored in existing schema objects. These commands are used to perform certain operations such as insertion, deletion, updation, and retrieval of the data from the database.
- These commands deal with the user requests as they are responsible for all types of data modification. The DML commands that deal with the retrieval of the data are known as Data Query language.
  NOTE: The DML commands are not auto-committed i.e., the changes and modifications done via these commands can be rolled back.
- **Some of the DML commands are:**

a) INSERT:-

- It is used to insert new rows into the table.
- ***To insert values according to the table structure:***
  INSERT INTO table_name
  VALUES value1, value2, value3;
- ***To insert values based on the columns:***
  INSERT INTO table_name column1, column2, column3
  VALUES value1, value2, value3;

Note:- While using the INSERT command, make sure that the datatypes of each column match with the inserted value, as this may lead to unexpected scenarios and errors in the database.

b) UPDATE:-

- It is used to update existing column(s)/value(s) within a table.
- ***To update the columns of a table based on a condition (General UPDATE statement):***

```
UPDATE table_name
SET column_1 = value1,
   column_2 = value2,
```

column_3 = value3,
[WHERE condition]

Note: Here, the SET statement is used to set new values to the particular column, WHERE clause is used to select rows for which the columns are updated for the given table.

c) DELETE:-

- It is used to delete existing records from a table, i.e., it is used to remove one or more rows from a table.
- ***To delete rows from a table based on a condition:***

```
DELETE FROM table_name [WHERE condition];
```

Note: The DELETE statement only removes the data from the table, whereas the TRUNCATE statement also frees the memory along with data removal. Hence, TRUNCATE is more efficient in removing all the data from a table.

d) MERGE:-

- It is a combination of the INSERT, UPDATE, and DELETE statements.
- It is used to merge data from a source table or query-set with a target table based on the specified condition.
- ***To delete an object:***

```
MERGE INTO target_table_name
USING source_table_name
   ON <condition>
WHEN MATCHED THEN
   UPDATE <statements>
WHEN NOT MATCHED THEN
   INSERT <statements>
```

e) EXPLAIN PLAN:-

- It is used to display the sequence of operations performed by the DBMS software upon the execution of a DML statement.
- ***To display the execution plan of a query:***

```
EXPLAIN PLAN FOR
   QUERY;
```

3. Data Control Language (DCL)

- Data Control Language (DCL) is a set of special commands that are used to control the user privileges in the database system. The user privileges include ALL, CREATE, SELECT, INSERT, UPDATE, DELETE, EXECUTE, etc.
- We require data access permissions to execute any command or query in the database system. This user access is controlled using the DCL statements. These statements are used to grant and revoke user access to data or the database.
- DCL commands are transactional i.e., these commands include rollback parameters. These commands include:

a) Grant:-

- It is used to provide user access to the database or its objects.
- ***To grant user privilege to specified users on a database object:***

```
GRANT <privilege>
ON <object>
TO user1, user2;
```

b) Revoke:-

- It is used to revoke user access to the database system.
- *To revoke user privilege to specified users on a database object :*
  REVOKE <privilege>
  ON <object>
  FROM user1, user2;
  Note: In practical usage, instead of having a separate language for every operation, a combination of DDL, DML, and DCL is used as part of a single database language such as SQL.

4. Transaction Control Language (TCL)

- Transaction Control Language (TCL) is a set of special commands that deal with the transactions within the database.
- A transaction is a collection of related tasks that are treated as a single execution unit by the DBMS software. Hence, transactions are responsible for the execution of different tasks within a database.
- The modifications performed using the DML commands are executed or rollbacked with the help of TCL commands. These commands are used to keep a check on other commands and their effects on the database. These include:

a) COMMIT:-

- It is used to permanently save all the modifications are done (all the transactions) by the DML commands in the database. Once issued, it cannot be undone.
- DBMS software implicitly uses the COMMIT command before and after every DDL command to save the change permanently in the database.

```
COMMIT;
```

b) ROLLBACK:-

- It is used to undo the transactions that have not already been permanently saved (or committed) to the database.
- It restores the previously stored value, i.e., the data present before the execution of the transactions.
- ***To undo a group of transactions since last COMMIT or SAVEPOINT:***

```
ROLLBACK;
```

- ***To undo a group of transactions to a certain point:***

```
ROLLBACK TO savepoint_name;
```

c) SAVEPOINT:-

- It is used to create a point within the groups of transactions to save or roll back later.
- It is used to roll the transactions back to a certain point without the need to roll back the whole group of transactions.
- ***To create a SAVEPOINT:***

```
SAVEPOINT savepoint_name;
```

- ***To release a SAVEPOINT:***

```
RELEASE SAVEPOINT savepoint_name;
```

d) AUTOCOMMIT:-

- It is used to enable/disable the auto-commit process that commits each transaction after its execution.
- ***To enable the AUTOCOMMIT process:***

- SET AUTOCOMMIT = 1 ;
- ***To disable the AUTOCOMMIT process:*** SET AUTOCOMMIT = 0;

Query Languages

- Query means question. Query language is a language which is used to manipulate and to retrieve information from a database.
- A query language, also known as data query language or database query language (DQL).
- Query language can be divided into two types:-

1. Procedural language
2. Non-procedural language

1. Procedural query language

- In a procedural query language all the instructions of database are written in a specific order to retrieve a particular data from database.
- The data is retrieved in relation form.
- It gives a step by step process to obtain the result of the query.
- It uses operators to perform queries.
- For Example: Relational algebra
- Structure Query language (SQL) is based on relational algebra.

2. Non-Procedural language

- In the non procedural query language sequence of operation is not specified while retrieving the information from the database.
- Users only specify what information is to be retrieved.
- For Example: Relational Calculus
- Query by Example (QBE) is based on Relational calculus

1. Procedural query language

Relational Algebra

- In DBMS relational algebra is a procedural query language that can be used to retrieve a new relation from one or more relations in the database or can be used to manipulate the relations.
- Relational algebra is theoretical foundations for relational databases.
- Relational Algebra is the fundamental block for modern language SQL and modern Database Management Systems such as Oracle Database, Mircosoft SQL Server, IBM Db2, etc.
- Relational Algebra accepts a Relation as input and outputs another Relation.
- Queries in relational algebra are performed using operators. A binary or unary operator can be used.
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Types of Relational operation

In the relational algebra we can define the operators in two types: basic operations and derived operations

a) Basic operations:-

There are some basic unary operations:

1. Select(σ)
2. Projection(π)
3. Rename (ρ)

There are some basic binary operations:

4. Union operation (υ)
5. Set Difference (-)

6. Cartesian product(X)

b) Derived operations:-

There are some derived binary operations:

7. Division
8. Intersection(∩)
9. Join Operations
   - **9.1** Inner Join:
     - **a)** Theta Join:
     - **b)** EQUI join:
     - **c)** NATURAL JOIN (⋈)
   - **9.2** Outer join:
     - **a)** Left Outer Join(A ⟕ B)
     - **b)** Right Outer Join: (A ⟖ B)
     - **c)** Full Outer Join: (A ⟗ B)

Derived uniary operation:

- Self Join

a) Basic operations

Here we have considered a student table for explaining the operations.

Student relation (table)

Enrol_no.	F_name	L_Name	Course	Age	year
101	John	Clay	CSE	21	2010
102	Minty	Shay	EC	21	2012
103	Clark	Stark	ME	21	2012
104	Shon	Noah	CSE	20	2010
105	vicky	Singh	EC	22	2011
106	john	Gham	EC	20	2012

Basic Unary operations

1. **Select Operation:** It is used to retrieve tuples(rows) from the table when the given condition is satisfied. It is a unary operator means it requires only one operand.
   It is denoted by sigma (σ).
   Notation: $\sigma_{p(r)}$
   Where:
   σ is used for selection prediction
   r is used for relation
   p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relations can use as relational operators like =, ≠, ≥, <, >, ≤.
   Example:-
   1. **$\sigma_{course = "CSE"}$(Student)**
      Result – Selects tuples from student where course is CSE.

Enrol_no.	F_name	L_name	Course	Age	year
101	John	Clay	CSE	21	2010
104	Shon	noah	CSE	20	2010

2. **Projection operation:-** If we have to retrieve some specific attributes (coloumn) from the relation we use projection operator. It is also known as vertical partitioning as it separates the table vertically.

Duplicate rows are automatically eliminated, as relation is a set.

It is also a unary operator.

Notation : $\prod_{A1,A2,..An}(r)$

Where

$\prod$ is used to represent projection

r is used to represent relation

A1,A2…An is the attribute list

Examples:

1. **$\prod_{F\_Name}$(student)**

   Result:- – project all the tuples of F_Name attribute from student. It delete the duplicate tuples.

F_name
John
Minty
Clark
Shon
Vicky

3. **Rename Operation (ρ):-** The results of relational algebra are also relations but without any name. The rename operation is used to rename the output relation.

The rename-operation is denoted using a small Greek letter rho (ρ).

Notation1: ρ x (E)

Where

ρ is used to denote the rename operator.

E is the result of expression or sequence of operation.

X is the new name to the relation.

Notation2: $\rho_{x(A1,A2,...An)}$ (E)

If we want to rename the table name and attributes name both then this notation returns the result of expression E under the name X and with the attributes renamed to A1,A2,…An.

Notation3: $\rho_{(A1,A2,...An)}$ (E)

If we want to rename the attributes name only then this notation returns the result of expression E with the attributes renamed to A1,A2,…An.

Example:-

1. **$\rho_{Aspirant}$ (student)**
2. **$\rho_{Aspirant(name1,name2)}$ ($\prod_{F\_Name,L\_name,Age}$(student))**

Result: It project F_name, L_Name and Age attributes by renaming the student relation to Aspirant and attributes F_name, L_name to name1 and name2.

Aspirant relation

Name1	Name2	Age
John	Clay	21
Minty	Shay	21
Clark	Stark	21
Shon	Noah	20
vicky	Singh	22
john	Gham	20

Basic binary operation

Here we have considered a student table and an employee tablefor explaining the operations.

Student relation

Enrol_no.	F_name	L_Name	Course	Age	year
101	John	Clay	CSE	21	2010
102	Minty	Shay	EC	21	2012
103	Clark	Stark	ME	21	2012
104	Shon	Noah	CSE	20	2010
105	vicky	Singh	EC	22	2011
106	john	Gham	EC	20	2012

Employee relation

Emp_id	F_Name	L_Name	department
E001	Ravi	Singh	IT
E002	Sarry	Singh	Admin
E003	Manoj	Sinha	IT
E004	Jack	Sheen	HR
E005	Bella	Stark	HR
E006	Sunny	Gupta	Admin

4. **Union operation (U):-** It is a binary operation which means it includes more than one relation (table).
   - It is the same as union operation in set theory.

Example:-

1. $\prod_{L\_Name}$(STUDENT) ∪ $\prod_{L\_Name}$(EMPLOYEE)

Result:- It shows the last name of the all the students and employees and remove duplicate last names.

L_Name
Clay
Shay
Stark
Noah
Singh
Gham

5. **Set difference (-):-** It is the same set difference operation as in set theory.
   - It is a binary operation which means it includes more than one relation (table).

Example:-

a) $\prod_{L\_Name}$(STUDENT) - $\prod_{L\_Name}$(EMPLOYEE)

Result:- It shows the last name of the all the students which are not present in employee relation.

L_Name
Clay
Shay
Noah
Gham

6. **Cartesian Product (cross product):-** Cartesian product is basic binary operator.
   - It combines information of two different relations into one.
   - It is also called Cross Product or Cross Join.
   - **Notation:- R X S**

Where

R and S are 2 different relations.

Example:

R

A	B
1	4
2	1
3	2

S

C	D
2	4
3	2

1. R X S:-

Result:- Each tuple of R is combined with each tuple of S.

A	B	C	D
1	4	2	4
1	4	3	2
2	1	2	4
2	1	3	2
3	2	2	4
3	2	3	2

b) Derived Operations: It is also known as extended operation. These operations can be derived from basic operations.

7. Division operations(÷):-

- Division Operation is represented by "division"(÷ or /) operator.
- Division can be expressed in terms of Cross Product, Set-Difference and Projection.
- It is used in queries that involve keywords "every", "all" like “at all”, “for all” or “in all”, “at every” or “in every” etc.

8. <u>Set Intersection (∩) Operator</u>:-

- Suppose R and S is two relations. The Set Intersection operation selects all the tuples that are common in both relations R & S.
- Set intersection is commutative **R∩S= S∩R**

9. <u>Join Operator:-</u>

- Joins are Additional / Derived operator from cartesian product which simplify the queries by using some conditions, however it does not add any new power to the basic relational algebra.
- Cartesian product of two relations (A x B) gives us all the possible tuples which may return a huge amount of data.
- Join is a combination of a two basic operators: Cartesian product and selection process.
- A Join operation combines two tuples from different relations, if and only if a given condition is satisfied.
- **Notation: A⋈ B.**

  Where A and B are two different tables.
- Join = Cartesian Product + Selection operator

$A \bowtie_C B = \sigma_C(A \times B)$

1. **Inner join:-** Inner join combines only those records from two tables that satisfy the matching condition.

- **We have main three types of inner joins:-**

a. **Natural join(A ⋈ B):-** Natural Join is by default inner join because the tuples which does not satisfy the conditions of join does not appear in result set.

- If there is at least one common attribute (column) exist between two relations, we can perform Natural join.
- The attributes must have the same name and domain.
- Natural join does not use any comparison operator but it is same as the Equi join.
- It implicitly compares all the common attributes (columns) in both relations.
- The main difference between the Equi join and Natural join is that common attributes appears only once in Natural join.
- The result of natural join is the set of all combinations of tuples in two relations A and B that are identical on their common attribute names.
- **Note:** Projection operation to Equi join of two relations results the Natural join.
- In terms of basic operators: **Natural Join = Projection + Selection + Cartesian product.**

  $\Pi_{(A.column1, A.coloumn2, B.coloumn2,.....)}(\sigma_{(A.column1=B.coloumn1)}(A \times B))$

- All distinct coloumns of A and B are in result.
- **Notation:- A ⋈ B**

  Where A and B are two tables**.**

Example:-In the above Student and Employee tables, select students whose roll_no is equal to emp_no of employees.

Student

Roll_No.	Name
1	A
2	B
3	C
4	D

Marks

Roll_No.	Marks
2	50
3	60
4	80
6	40

Student $\bowtie_{student.roll\_no= Marks.Roll\_no}$ **Employee**

Roll_no.	Name	Marks
2	A	50
3	B	60
4	C	80

b. **Theta join:-** It is also called conditional join.

- Theta Join uses all kinds of comparison operators(>,<,>= <=,=,≠).
- In terms of basic operators: **Theta join = selection operator + cross product**

  $\sigma_{(A.column>B.\ column)}(A \times B)$

- **Notation:- A $\bowtie_\theta$ B**

Example1:- Suppose we have two tables student and Employee and we have to join the tables where student.roll_no< Employee.Emp_no.

Student

Roll_no.	Name
1	A
2	B
3	C
4	D

Marks

Roll_no.	Marks
2	50
3	60
4	80
6	40

Student $\bowtie$ **student.roll_no> Marks.roll_no** Employee

Roll_no.	Name	Roll_No.	Name
3	C	2	50
4	D	2	50
4	D	3	60

c. Equi join:-

- Equijoin is a special case of Theta join where we check only equality condition between two attributes.
- Theta join with equivalence (=) condition becomes Equi join.
- If the values of two attributes will be equal in result of equijoin, only one attribute will be appeared in result.
- In terms of basic operators: **Equi Join = projection + selection + cross product**
- $\Pi_{(A.column1, A.coloumn2, B.coloumn2,\ldots.)}(\sigma_{(A.column1=B.coloumn1)}(A\times B))$
- Only coloumn1 will come once in the result.
- **Notation:-** $(A) \bowtie_{A.column = B.column} (B)$

Where A and B are two tables**.**

Example:-In the above Student and Employee tables, select students whose roll_no is equal to emp_no of employees.

Student $\bowtie$ **student.roll_no= Employee.Emp_no** Employee

Student

Roll_no.	Name
1	A
2	B
3	C
4	D

Employee

Emp_no.	Name
2	X
3	P
4	N
6	H

Roll_no.	Name	Emp_no.	Name
2	B	2	X
3	C	3	P
4	D	4	N

2. Outer join:- The outer join operation is an extension of the join operation that avoids loss of information.

- Inner Join includes records with matching attributes and the rest are discarded in the resulting relation.
- But in outer joins all the rest of the tuples from the given relations also included in the resulting relation.
- Outer Join contains matching tuples that satisfy the matching condition, along with some or all tuples that do not satisfy the matching condition.
- It is based on both matched or unmatched records.
- We can also use some condition with outer join.
- **There are three types of outer joins:-**

a. Left outer join(A ⟕ B):- In the left outer join, resulting table holds all records in common from both relations and all records from the left relation.

- However, if there is no matching tuple is found in right relation, then we fill NULL value for that particular record.
- **Notation:- (A ⟕ B)**

Example:- Refer the above student and Employee table to perform left outer join.

Student ⟕ Employee

Roll_no.	Name	Emp_no.	Name
1	A	NULL	NULL
2	B	2	B
3	C	3	C
4	D	4	D

b. Right outer join (A ⟖ B):- In the right outer join, resulting table holds all records in common from both relations and all records from the right relation.

- However, if there is no matching tuple is found in left relation, then we fill NULL value for that particular record.
- We can also use some condition with right outer join.
- **Notation:- (A ⟖ B)**

Example:- Refer the above student and Employee table to perform right outer join.

Student ⟖ Employee

Roll_no.	Name	Emp_no.	Name
2	B	2	B
3	C	3	C
4	D	4	D
NULL	NULL	9	E

c. Full outer join(A ⟗ B):- In Full outer join, all the tuples from both Left relation and right relation are included in the resulting relation. The tuples of both relations which do not satisfy join condition contain NULL with their respective unmatched attributes.

Example:- Refer the above student and Employee table to perform right outer join.

Student ⟗ Employee

Roll_no.	Name	Emp_no.	Name
1	A	NULL	NULL
2	B	2	B
3	C	3	C
4	D	4	D
NULL	NULL	9	E

3. **Self join:-** A self-join is a join that can be used to join a table with itself. Hence, it is a unary relation. Especially when the relation has a foreign key which references to its own primary key.

- To join a relation to itself means that each tuple of the relation is combined with itself and with every other tuple of the relation.
- It can be viewed as a join of two copies of the same relation.

Example:- To use self join in Student table where first student table roll_no>Second student table roll_no
First we rename the two student table as S1 and S2 **ρ_{S1}(student) and ρ_{S2}(student)**

ρ_{S1}(student) $\bowtie_{S1.Roll\_no>S2.Roll\_no}$ ρ_{S2}(student)

S1

Roll_no.	Name
1	A
2	B
3	C
4	D

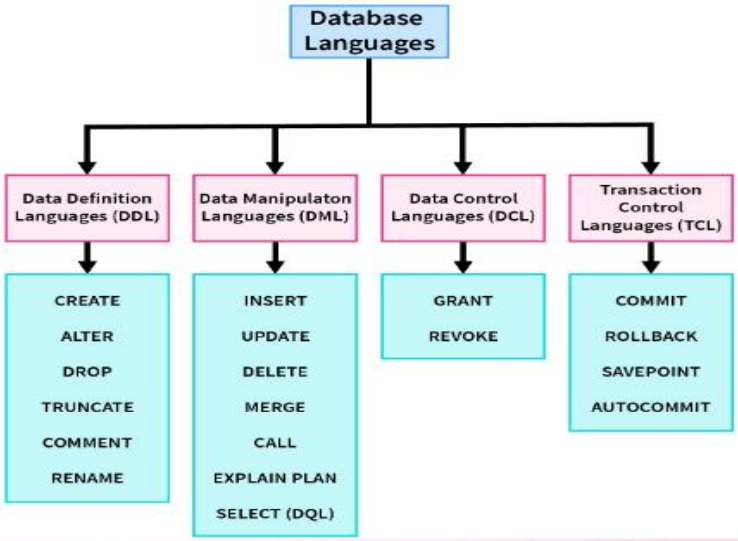
S2

Roll_no.	Name
1	A
2	B
3	C
4	D

S1.Roll_no.	S1.Name	S2.Roll_no.	S2.Name
2	B	1	A
3	C	1	A
3	C	2	B
4	D	1	A
4	D	2	B
4	D	3	C

Database Language(Structured Query Language)

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).



There are 4 types of database languages:-

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Control Language (DCL)
4. Transaction Control Language (TCL)
5. Data Query Language (DQL)

1. Data Definition Language (DDL)

- Data Definition Language (DDL) is a set of special commands that allows us to define and modify the structure and the metadata of the database.
- These commands can be used to create, modify, and delete the database structures such as schema, tables, indexes, etc.
- Every change implemented by a DDL command is auto-committed (the change is saved permanently in the database), these commands are normally not used by an end-user
- Some of the DDL commands are:

a) CREATE:-

- It is used to create the database or its schema objects.
- To create a new database:
  CREATE DATABASE database_name;
- To create a new table:
  CREATE TABLE table_name (
  column_1 DATATYPE,
  column_2 DATATYPE,
  column_n DATATYPE);

Example 1:

CREATE TABLE Emp(Emp_ID int, Name varchar(20), Age int, Address varchar(100), Salary numeric(10,2));

Emp_Id	Name	Age	Address	Salary

b) DROP:-

- It is used to delete the database or its schema objects.
- To delete an existing table:
  DROP TABLE table_name;
- To delete the whole database:
  DROP DATABASE database_name;

c) ALTER:-

- It is used to modify the structure of the database objects.
- To add new column(s) in a table:

```
ALTER TABLE table_name ADD (
  column_1 DATATYPE,
   column_2 DATATYPE,
   column_n DATATYPE );
```

- To change the datatype of a column in a table:

```
ALTER TABLE table_name
MODIFY column_name DATATYPE;
```

- To remove a column from a table:

```
ALTER TABLE table_name  DROP COLUMN column_name;
```

d) TRUNCATE:-

- It is used to remove the whole content of the table along with the deallocation of the space occupied by the data, without affecting the table's structure.
- To remove data present inside a table:

TRUNCATE TABLE table_name;

NOTE:- We can also use the DROP command to delete the complete table, but the DROP command will remove the structure along with the contents of the table. Hence, if you want to remove the data present inside a table but not the table itself, you can use TRUNCATE instead of DROP.
Example:
TRUNCATE TABLE Emp;

Emp Table before truncate command:-

Emp_Id	Name	Age	Location	Salary
102	A	25	Gurgoan	20000
103	B	34	Delhi	25000

Emp Table after truncate command:-

Emp_Id	Name	Age	Location	Salary

e) COMMENT:-

- It is used to add comments about the tables, views, and columns into the data dictionary. These comments can help the developers to better understand the structure of the database.
- To comment on table/view:

```
COMMENT ON TABLE table_name  IS 'text';
```

- To comment on a column:
  COMMENT ON COLUMN table_name.column_name IS 'text';
- To drop a comment:
  COMMENT ON TABLE table_name IS ' ';

f) RENAME:-

- It is used to change the name of an existing table or a database object.
- To rename a table:
  RENAME old_table_name TO new_table_name;

- To rename a column of a table:
  ALTER TABLE table_name
  RENAME COLUMN old_Column_name to new_Column_name;

Example:
RENAME TABLE Emp To Employee;

2. Data Manipulation Language (DML)

- Data Manipulation Language (DML) is a set of special commands that allows us to access and manipulate data stored in existing schema objects.
- These commands are used to perform certain operations such as insertion, deletion, updation, and retrieval of the data from the database.
- These commands deal with the user requests as they are responsible for all types of data modification.
  NOTE: The DML commands are not auto-committed i.e., the changes and modifications done via these commands can be rolled back.

- Some of the DML commands are:

a) INSERT:- It is used to insert new rows into the table.

- To insert values according to the table structure:
  INSERT INTO table_name
  VALUES value1, value2, value3;
- To insert values based on the columns:
  INSERT INTO table_name column1, column2, column3
  VALUES value1, value2, value3;

Note:- While using the INSERT command, make sure that the datatypes of each column match with the inserted value, as this may lead to unexpected scenarios and errors in the database.

b) UPDATE:- It is used to update existing column(s)/value(s) within a table.

- To update the columns of a table based on a condition (General UPDATE statement):
  UPDATE table_name
  SET column_1 = value1,
  column_2 = value2,
  column_3 = value3,
  [WHERE condition]

c) DELETE:- It is used to delete existing records from a table, i.e., it is used to remove one or more rows from a table.

- To delete rows from a table based on a condition:

```
DELETE FROM table_name [WHERE condition];
```

Note: The DELETE statement only removes the data from the table, whereas the TRUNCATE statement also frees the memory along with data removal. Hence, TRUNCATE is more efficient in removing all the data from a table.

d) MERGE:-

- It is a combination of the INSERT, UPDATE, and DELETE statements.
- It is used to merge data from a source table or query-set with a target table based on the specified condition.
- To delete an object:

```
MERGE INTO target_table_name
USING source_table_name
   ON <condition>
WHEN MATCHED THEN
   UPDATE <statements>
WHEN NOT MATCHED THEN
   INSERT <statements>
```

3. Data Control Language (DCL)

- Data Control Language (DCL) is a set of special commands that are used to control the user privileges in the database system. The user privileges include ALL, CREATE, SELECT, INSERT, UPDATE, DELETE, EXECUTE, etc.
- These statements are used to grant and revoke user access to data or the database.
- These commands include:

a) Grant:-

- It is used to provide user access to the database or its objects.
- To grant user privilege to specified users on a database object:

```
GRANT <privilege>
ON <object>
TO user1, user2;
```

b) Revoke:-

- It is used to revoke user access to the database system.
- To revoke user privilege to specified users on a database object :

```
REVOKE <privilege>
ON <object>
FROM user1, user2;
```

4. Transaction Control Language (TCL)

- Transaction Control Language (TCL) is a set of special commands that deal with the transactions within the database.
- A transaction is a collection of related tasks that are treated as a single execution unit by the DBMS software.

- The modifications performed using the DML commands are executed or rollbacked with the help of TCL commands.
- These commands are used to keep a check on other commands and their effects on the database.

a) COMMIT:-

- It is used to permanently save all the modifications are done (all the transactions) by the DML commands in the database. Once issued, it cannot be undone.
- DBMS software implicitly uses the COMMIT command before and after every DDL command to save the change permanently in the database.

Example:

```
DELETE FROM Emp  WHERE age = 25;
COMMIT;
```

b) ROLLBACK:-

- It is used to undo the transactions that have not already been permanently saved (or committed) to the database.
- It restores the previously stored value, i.e., the data present before the execution of the transactions.
- To undo a group of transactions since last COMMIT or SAVEPOINT:

```
ROLLBACK;
```

- To undo a group of transactions to a certain point:

```
ROLLBACK TO savepoint_name;
```

c) SAVEPOINT:-

- It is used to create a point within the groups of transactions to save or roll back later.
- It is used to roll the transactions back to a certain point without the need to roll back the whole group of transactions.

- To create a SAVEPOINT:

```
SAVEPOINT savepoint_name;
```

- To release a SAVEPOINT:

```
RELEASE SAVEPOINT savepoint_name;
```

d) AUTOCOMMIT:-

- It is used to enable/disable the auto-commit process that commits each transaction after its execution.
- To enable the AUTOCOMMIT process:

```
SET AUTOCOMMIT = 1 ;
```

- To disable the AUTOCOMMIT process:

```
SET AUTOCOMMIT = 0;
```

5. Data Query Language(DQL)

- DQL is used to fetch the data from the database.
- DQL has only one command:

a) SELECT:-

- This is the same as the projection operation of relational algebra.
- SELECT statement is used to select a set of data from a database table or simply SELECT statement is used to retrieve data from a database table.
- It returns data in the form of a result table. These result tables are called result-sets.
- SELECT statement specifies column names, and FROM specifies table name
- SELECT command is used with different Conditions and CLAUSES.

Syntax:-

SELECT column_name1, column_name2, column_name3, ... column_nameN
FROM table_name;

Example:- SELECT Name, Address, FROM Emp;

Name	Address
Ram	Delhi
Kamal	Pune
Kajal	Chennai
Mahi	Patna

To retrieve all the fields from the table we use asterisk *:

Example:-

SELECT * FROM Emp;

Emp_id	Name	Age	Address	Salary
1	Ram	32	Delhi	20000
2	Kamal	25	Pune	15000
3	Kajal	22	Chennai	45000
4	Mahi	24	Patna	10000

Clauses used with Select statement

- A clause in SQL is a built-in function that helps to filter and analyze the queries from a database table.
- A clause receives a conditional expression, i.e. a column name or some terms involving the columns.

This is a reference table for given queries.

Emp_id	Name	Age	Address	Salary	Dept_id
1	Ram	32	Delhi	20000	300
2	Kamal	25	Pune	15000	300
3	Kajal	22	Delhi	45000	300
4	Mahi	24	Patna	10000	301
5	Vinod	28	Mumbai	20500	301
6	Ramesh	26	Bilaspur	13000	302
7	Dinesh	23	Mumbai	16000	303

a) Where:- The WHERE clause is used with SELECT statement to return only those rows from the table, which satisfy the specified condition in the query.

- WHERE Clause is used to specify a condition while fetching the data from a single table or by joining with multiple conditions
- The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement.

Syntax:

```
SELECT column1, column2, …. columnN
FROM table_name(s)
WHERE condition;
```

Example:- to retrieve Emp_id, Name, from Emp table whose age is greater than 25.

SELECT Emp_Id, Name,Age, Salary **FROM** Emp **WHERE age>25**

Result:-

Emp_id	Name	Age	Salary
1	Ram	32	20000
5	Vinod	28	20500
6	Ramesh	26	13000

b) Group By:- The GROUP BY clause is used to group the result set of the rows that have the same values in the result set from the database tables.

- In SQL, the GROUP BY statement is used for organizing similar data into groups.
- **For Example: "find the number of customers in each country".**
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.
- GROUP BY clause is used with the SELECT statement in the SQL query.
- GROUP BY clause is placed after the WHERE clause in SQL.
- GROUP BY clause is placed before the ORDER BY clause in SQL.

Syntax:

SELECT column_name(s), aggregate_function(column_name) **FROM** table_name **[WHERE condition]**
GROUP BY column_name(s) **[ORDER BY** column_name(s)**];**

Example1:- Write a query to find the number of employee in each department

SELECT Dept_id, COUNT(*) **FROM** Emp **GROUP BY** Dept_id

Dept_id	Count(*)
300	3
301	2
302	1
303	1

Example2:- Write a query to find the number of employee in each department, sorted low to high
Query:- SELECT Dept_id, COUNT(*) **FROM** Emp **GROUP BY** Dept_id **ORDER BY** COUNT(*);

Dept_id	Count(*)
302	1
303	1
301	2
300	3

c) Having:- HAVING clause can be used in a GROUP BY clause. It is used to filter the groups created by the GROUP BY clause in the database tables.

- The HAVING clause enables you to specify conditions that filter which group results appear in the results.
- The HAVING clause was added to SQL because the aggregate functions like SUM, AVG, MIN, MAX,COUNT cannot be used with WHERE clause.
- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.
- In SQL query, HAVING clause is placed after the GROUP BY clause

Syntax:

SELECT column_name(s), **aggregate_function(**column_name) **FROM** table_name
[**WHERE** condition] **GROUP BY** column_name(s) **HAVING** condition [**ORDER BY** column_name/s)];

Example1:- Write a query to display departments where the number of employee is greater than 2
Query:- SELECT Dept_id, COUNT(*) **FROM** Emp **GROUP BY** Dept_id **HAVING** COUNT(*) >1;

Dept_id	Count(*)
300	3
301	2

Example2:- Write a query to find the department for which the sum of salaries of employees is more than 10000.
Query:- SELECT Dept_ID, **SUM**(Salary) **FROM** Emp **GROUP BY** Dept_id **HAVING** SUM(Salary)>20000;

Dept_id	Sum(Salary)
300	80000
301	30500

Example3:- Write a query to find the sum of salaries of employee that lives at same location
Query:- SELECT Address, **SUM**(Salary) **FROM** Emp **GROUP BY** Address **HAVING** COUNT(Address)>=2;

Address	Sum(Salary)
Delhi	65000
Mumbai	36500

d) Order By:- The ORDER BY clause in SQL is used for sorting the records of the database tables.

- The output of the SELECT queries do not have any specific order. The ORDER BY Clause allows us to specify order for the query output.
- ORDER BY Clause is used with SELECT statement for arranging retrieved data in sorted order.
- The ORDER BY Clause by default sorts the retrieved data in ascending order [ASC].
- To sort the data in descending order, DESC keyword is used with ORDER BY clause.

- In ORDER BY Clause multiple columns can be used Where:-
  - 1st column is used as primary ordering field.
  - 2nd column is used as secondary ordering field and so on.

Syntax:
SELECT column-list
FROM table_name
[**WHERE** conditions]
ORDER BY column1, column2, .. columnN [ASC/ DESC];

Example1:- To display details of Emp table in decending order by salary
Query:- SELECT Emp_id, Name, Age,Salary **FROM** Emp **ORDER BY** Salary **DESC**;

Emp_id	Name	Age	Salary
3	Kajal	22	45000
5	Vinod	28	20500
1	Ram	32	20000
7	Dinesh	23	16000
2	Kamal	25	15000
6	Ramesh	26	13000
4	Mahi	24	10000

Example2:- To sort details of Emp table in ascending order by name and salary.
Query:- SELECT Emp_id, Name, Age, Salary **FROM** Emp **ORDER BY** Name, Salary

Emp_id	Name	Age	Salary
7	Dinesh	23	16000
3	Kajal	22	45000
2	Kamal	25	15000
4	Mahi	24	10000
1	Ram	32	20000
6	Ramesh	26	13000
5	Vinod	28	20500

SQL Operators

- SQL operators are used with the WHERE clause to make more precise conditions for fetching data from database by combining more than one condition together.
- There are some following SQL operators:
  1. SQL Arithmetic Operators
  2. SQL Comparison operator
  3. SQL Logical Operator
  4. SQL Special Operator

Emp Table

Emp_id	Name	Age	Address	Salary	Dept_id
1	Ram	32	Delhi	20000	300
2	Kamal	25	Pune	15000	300
3	Kajal	22	Delhi	45000	300
4	Mahi	24	Patna	10000	301
5	Vinod	28	Mumbai	20500	301
6	Ramesh	26	Bilaspur	13000	302
7	Dinesh	23	Mumbai	16000	303

a) SQL Arithmetic Operators

Arithmetic operations are +,-,*,/,% etc.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Example:- To retrieve Emp_ID, Name, Salary plus 3000 from Emp table

Query:- SELECT Emp_ID, Name, Salary + 3000 **FROM** Emp;

Emp_id	Name	Salary
1	Ram	23000
2	Kamal	18000
3	Kajal	48000
4	Mahi	13000
5	Vinod	23500
6	Ramesh	16000
7	Dinesh	19000

b) SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to

Example:- To retrieve Emp_id, Name, from Emp table whose salary is greater than 20000.
Query:- SELECT Emp_Id, Name,Salary **FROM** Emp **WHERE** Salary>20000

Emp_id	Name	Salary
3	Kajal	45000
5	Vinod	20500

c) SQL Logical Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- To make more precise conditions for fetching data from database by combining more than one condition together.
- The AND and OR operators are used to filter records based on more than one condition.
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE

Operator	Description
AND	TRUE if all the conditions separated by AND is TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
NOT	Displays a record if the condition(s) is NOT TRUE. It reverse the meaning of any operator with which it is used. This is a negate operator. Eg: NOT EXISTS, NOT BETWEEN, IS NOT NULL, NOT IN etc.

a) AND operator:- AND operator (or AND condition) is used to combine multiple conditions with WHERE clause, in SELECT, UPDATE or DELETE statements.
All conditions must be satisfied for a record to be selected.

Syntax:
SELECT column 1, column2, ... columnN
FROM table_name
WHERE condition1 **AND** condition2.....**AND** conditionN;

Example1: To retrieve all records from Emp table whose salary is greater than 2000 and age is less than 25
Query:- SELECT Emp_id,Name,Age, Salary **FROM** Emp **WHERE** Salary>2000 AND Age<25;

Emp_id	Name	Salary	Age
3	Kajal	45000	22

b) OR operator:- OR operator (or OR condition) is used to combine multiple conditions with WHERE clause, in SELECT, UPDATE or DELETE statements.
Any one of the conditions must be satisfied for a record to be selected.

Syntax:
SELECT column1, column2, columnN ...
FROM table_name
WHERE condition1 OR condition2.....OR conditionN;

Example:- Select all records from Emp table whose salary is greater than 20000 or whose age is less than 25
Query:- SELECT Emp_id, Name, Salary **FROM** Emp **WHERE** Salary>20000 OR Age<25;

Emp_id	Name	Age	Salary
3	Kajal	22	45000
4	Mahi	24	10000
5	Vinod	28	20500
7	Dinesh	23	16000

c) **NOT operator:-** NOT operator (or NOT condition) is used to negate the conditions with the WHERE clause, in SELECT, UPDATE or DELETE statements.
At displays a record if the condition(s) is NOT TRUE.
NOT reverses the meaning of any operator with which it is used. This is a negate operator.
Syntax:-
SELECT column1, column2,……. columnN
FROM table_name
WHERE NOT condition;

Example:- Selects all records from where address is not Mumbai
Query:- SELECT Emp_Id, Name**,**Address **FROM** Emp **WHERE** NOT Address="Mumbai';

Emp_id	Name	Address
1	Ram	Delhi
2	Kamal	Pune
3	Kajal	Delhi
4	Mahi	Patna
6	Ramesh	Bilaspur

Example:- Selects all records from Emp table whose salary is greater than 20000 and age is less than 25 or whose Emp ID is 2.
Query:- SELECT Emp_Id, Name**,** Age, Salary **FROM** Emp **WHERE** (Salary>20000 **AND** Age<25) **OR** (Emp_ID=2);

Emp_id	Name	Age	Salary
2	Kamal	25	15000
3	Kajal	22	45000

d) SQL Special Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
DISTINCT	It remove duplicates values from results set
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
SOME	TRUE if any of the subquery values meet the condition
IS NULL	TRUE if the row is NULL
LIMIT/TOP	Return the upper limit on the number of tuples

Note:- ANY and SOME are the same. You can use any one.

a) ALL operator:- It returns a boolean value as a result.

- It returns TRUE if ALL of the subquery values meet the condition.
- It is used with SELECT, WHERE and HAVING statements.
- ALL operator is used to return all records of the SELECT statement.
- The result of the ALL operator is true only if the condition satisfies all values in the range.
- When the subquery does not return any row, ALL operator will return true.
- **Greater than (>) ALL means** greater than the maximum value. **For example,** greater than ALL (5, 50, 80) means larger than 80 as the numbers greater than it will be greater than others.
- **Less than (<) ALL means** less than the minimum value. **For example,** Less than (<) ALL (16,25,9,2,28) means less than the minimum value that is 2 as the numbers less than the minimum are less than others.
- Standard comparison operator(=, <>, !=, >, >=, <, or <=) are used with ALL operator.

Syntax With SELECT:-
SELECT ALL column_name(s) **FROM** table_name **WHERE** condition;

Syntax With WHERE or HAVING:-
SELECT column_name(s) **FROM** table_name
WHERE column_name **ALL** (**SELECT** column_name **FROM** table_name **WHERE** condition);

Example1:- To get the name of all Employees from the Emp table.
Query:- SELECT ALL Name FROM Emp WHERE TRUE;
Result:-

Name
Ram
Kamal
Kajal
Mahi
Vinod
Ramesh
Dinesh

Example3:- To get the name of all Employees from the Emp table where emp_id is less than all head id where dept_id is not equal to 300 in department table.

Query:- SELECT Name FROM Emp WHERE Emp_id <ALL (SELECT Emp_id FROM Department WHERE Dept_id <> 300);

Result:- SELECT Name FROM Emp WHERE Emp_id <ALL (4,6)

Name
Ram
Kamal
Kajal

NOT ALL Operator:- It is opposite to ALL operator.

b) Distinct (unique):- The SELECT DISTINCT statement is used to return only distinct (different) values from the table or it is used to return only unique values.

- DISTINCT Clause is used to remove duplicates from results set of a SELECT statement.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.
- When only one column (expression) is provided in the DISTINCT clause, the query will return the unique values for that column.
- When more than one column (expression) is provided in the DISTINCT clause, the query will retrieve unique combinations for the columns listed.
- In SQL, the DISTINCT clause doesn't ignore NULL values. So when using the DISTINCT clause in your SQL statement, your result set will include NULL as a distinct value.

Syntax:

SELECT DISTINCT column1, column2, ... columnN
FROM table_name;

Example1:-

Without DISTINCT keyword

SELECT Address **FROM** Emp

Address
Delhi
Pune
Delhi
Patna
Mumbai
Bilaspur
Mumbai

With DISTINCT Keyword

SELECT DISTINCT Address **FROM** Emp

Address
Delhi
Pune
Patna
Mumbai
Bilaspur

c) Like Operator:- The LIKE operator is used in the WHERE condition to filter data based on some specific pattern.

- It can be used with numbers, string, or date values. However, it is recommended to use the string values.
- LIKE operator is used for operating the wildcards on our database.
- **Two wildcards used in conjuction with the Like Operator are as follows:**

➢ **[%] :-** Represents zero, one, or multiple characters.
➢ **[_] :-** Represents one single character.

Syntax:-
SELECT Column1, column2,……. **FROM** table_name **WHERE** column **LIKE** pattern;

Example1:- Select all employess with a name starting with "R"
Query:- SELECT * **FROM** Emp **WHERE** Name **LIKE** "R%";
Result:-

Emp_id	Name	Age	Address	Salary	Dept_id
1	Ram	32	Delhi	20000	300
6	Ramesh	26	Bilaspur	13000	302

LIKE operator with wildcard operator (%, _)

Pattern	Description
Where Name LIKE 'john'	Returns records whose Name value is 'john' or 'JOHN' or 'John' or 'jOhN' or 'JoHn'.
Where Name LIKE 'j%'	Returns records whose Name value starts with 'j' or 'J' followed by any number of characters or numbers. Ex:- 'Jay', 'Jayesh' etc
Where Name LIKE '%a'	Returns records whose Name value contains 'a' or 'A' at last position Or return the records whose name ends with 'a' or 'A'. Ex:- 'Radha', 'Sadhna' etc
Where Name LIKE '%r%'	Returns record whose Name value contains 'r' or 'R' at any position. 'Suryank', 'Rubi' etc.
Where Name LIKE 'r%h'	Returns records whose Name value should start with 'r' or 'R' and ends with 'h' or 'H'. Ex:- 'Rakesh', 'Ramesh' etc.
Where Name LIKE '___a'	Returns record whose Name value contains four characters and the fourth character must be 'a' or 'A'. E.g. 'Maya', 'Jaya' etc.
Where Name LIKE '_a%'	Returns records whose Name value contains the second characters 'a' or 'A'. Ex:- 'Ramesh', 'Aarav' etc.
Where Name LIKE '%s_'	Returns records whose Name value has the second last character is either 's' or 'S'. Ex:- 'Suresh', 'Ramesa' etc
Where Name LIKE '___'	Returns records whose Name value must be three characters long. Ex:- 'Jay', 'Ron' etc.
Where Name LIKE '___%'	Returns records whose Name value must contain at least three characters or more. Ex:- 'Rohit', 'Prerna' 'Sona' etc.
Where Name LIKE 'A[i,m,t]'	Returns records whose Name value starts from 'A' and followed by any characters specified in []. Ex:- 'Ajit', 'Amar', 'Amrit' etc.
FirstName LIKE 'A[^i,m,t]'	Returns records whose FirstName value starts from 'A' and should not followed by any characters specified in []. Ex:- 'Ajay', 'Ananya' etc

d) In Operator:- The IN operator allows us to specify multiple values in a WHERE clause.

- It allows us to easily test if an expression matches any value in a list of values.
- The IN operator is a shorthand for multiple OR conditions. So it used to replace multiple OR conditions.
- In the case of nested queries IN operator first executes inner query(subquery) and then it takes inner query result for outer query.

Syntax: SELECT column_name(s) **FROM** table_name **WHERE** column_name **IN**(value1, value2, ... valueN);

OR (In Subqueries)

SELECT column_name(s) **FROM** table_name **WHERE** column_name **IN(SELECT STATEMENT);**

Subquery:- A subquery is a query which is written in where expression of another query.

Example 1: Selects all Employees located in Delhi, Bilaspur and Patna
Query:- SELECT Name, age, address **FROM** Emp **WHERE** Address **IN ('Delhi', 'Pune',' Mumbai')**
Result:-

Name	Age	Address
Ram	32	Delhi
Kamal	25	Pune
Kajal	22	Delhi
Vinod	28	Mumbai
Dinesh	23	Mumbai

e) BETWEEN Operator:- BETWEEN operator (BETWEEN...AND) is used to select values within given range.(give minimum and maximum values).

The values can be numbers, text, or dates.
The BETWEEN operator is inclusive i.e. start and end values are included.
Syntax:
SELECT column_name(s) **FROM** table_name **WHERE** column_name **BETWEEN** value1 **AND** value2;

Example1: Selects all Employees with the Salary between 15000 to 25000
Query:- SELECT Name, Address, Salary **FROM** Emp **WHERE** Salary **BETWEEN** 15000 **AND** 25000;
Result:-

Name	Address	Salary
Ram	Delhi	20000
Kamal	Pune	15000
Vinod	Mumbai	20500
Dinesh	Mumbai	16000

SQL Function

SQL functions are divided into two categories:

1. Aggregate Functions
2. Scalar Functions

1. Aggregate Functions

- An aggregate function performs a calculation on multiple values and returns a single value.
- SQL provides many built-in functions that can be used to compute multiple values and return the required result.
- It is also used to summarize the data.
- An aggregate function ignores NULL values when it performs the calculation, except for the count(*) function.
- Some aggregate function are as follows:

1. COUNT()
2. SUM()
3. AVG()
4. MAX()
5. MIN()
6. FIRST()
7. LAST()

Count():-

Syntax:

SELECT COUNT(column_name) FROM table_name WHERE condition;

COUNT(*) :- It returns the total number of rows in a given table including null.

Or

COUNT(column_name):- It returns the total number of non-null values present in the column which is passed as an argument in the function.

Or

COUNT(DISTINCT column_name):- It returns the total number of distinct non-null values present in the column

SUM() Function:- The SUM() function takes the name of the column as an argument and returns the sum of all the non NULL values in that column.

Avg Function:- The AVG() aggregate function takes the name of the column as an argument and returns the average of all the non NULL values in that column.

SUM and Avg works only on numeric fields. For non numeric (i.e string) column it returns 0.

MAX() returns NULL when no row is selected.

MIN() returns NULL when no row is selected.

FIRST() Function:- The FIRST() function returns the first value of the selected column.

LAST() Function:- The LAST() function returns the last value of the selected column.

2. Scalar functions

- Aggregate functions and Scalar functions both return a single value.
- Aggregate functions operate on many records while Scalar functions operate on each record independently.
- Scalar functions are as follows:-

1. UCASE()
2. LCASE()
3. MID()

4. LEN()
5. ROUND()
6. NOW()
7. FORMAT()

1. UCASE(): It converts the value of a field to uppercase.

Syntax:

SELECT UCASE(column_name) FROM table_name;

Example:- Converting names of employees from the table emp to uppercase.

Queries:- SELECT UCASE(name) FROM Emp;

Result:-

Name
RAM
KAMAL
KAJAL
MAHI
VINOD
RAMESH
DINESH

2. LCASE():- It converts the value of a field to lowercase.

Syntax:

SELECT LCASE(column_name) FROM table_name;

Example:- Converting names of employees from the table Emp to lowercase.

Queries:- SELECT LCASE(Name) FROM Emp;

Result:-

Name
ram
kamal
kajal
mahi
vinod
ramesh
dinesh

3. MID() function:- The MID() function extracts texts from the text field.

Syntax:

SELECT MID(column_name,start,length) AS some_name FROM table_name;

Note:- Specifying length is optional here, and start signifies start position (starting from 1)

Example:- Fetching first four characters of names of employees from the Emp table.

Query:- SELECT MID(name,1,4) FROM Emp;

Result:-

Name
Ram
Kama
Kaja
Mahi
Vino
Rame
Dine

4. **LEN() function:** The LEN() function returns the length of the value in a text field.
   Syntax:
   SELECT LENGTH(column_name) FROM table_name;

Example:- Fetching length of names of employees from Emp table.
Query:- SELECT LENGTH(NAME) FROM Students;
Result:-

Name
3
5
5
4
5
6
6

5. **ROUND():-** The ROUND() function is used to round a numeric field to the number of decimals specified.
NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 gets rounded off to 6.
Syntax:
SELECT ROUND(column_name,decimals) FROM table_name;
decimals:- number of decimals to be fetched.

6. **NOW():-** The NOW() function returns the current system date and time.
Syntax:
SELECT column_name NOW() FROM table_name;

Example:- Fetching current system time.
Query:- SELECT name, NOW() AS DateTime FROM Emp;
Result:-

NAME	DateTime
RAM	7/28/2023 11:59:11 PM
KAMAL	7/28/2023 11:59:11 PM
KAJAL	7/28/2023 11:59:11 PM
MAHI	7/28/2023 11:59:11 PM
VINOD	7/28/2023 11:59:11 PM
RAMESH	7/28/2023 11:59:11 PM
DINESH	7/28/2023 11:59:11 PM

7. **FORMAT():-** The FORMAT() function is used to format how a field is to be displayed.
Syntax:-
SELECT FORMAT(column_name,format) FROM table_name;

Example:- Formatting current date as 'YYYY-MM-DD'.
Query:- SELECT Name, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Emp;

Result:-

Name	Date
RAM	2023-07-28
KAMAL	2023-07-28
KAJAL	2023-07-28
MAHI	2023-07-28
VINOD	2023-07-28
RAMESH	2023-07-28
DINESH	2023-07-28

Alias in SQL

- SQL ALIASES are used to give a temporary name for columns or tables.
- COLUMN ALIASES are used to make column names more readable by giving it a temporary name..
- An alias is created with the AS keyword.

Alias Column Syntax:-

SELECT column_name AS alias_name FROM table_name;

Alias Table Syntax:-

SELECT column_name(s) FROM table_name AS alias_name;

Example1:- SELECT Address as city From Emp as Employee

Result:- Employee

City
Delhi
Pune
Delhi
Patna
Mumbai
Bilaspur
Mumbai

Normalization in DBMS

Before studying normalization we must know about keys in RDBMS and functional dependency of attributes in RDBMS.

Functional Dependency

- The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.
- Functional Dependency (FD) is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS).
- A functional dependency is denoted by an arrow "→".
- The functional dependency of attribute X on attribute Y is represented by X → Y.
- The left side of FD is known as a determinant (X), the right side of the production is known as a dependent(Y).
- X and Y are single attribute or set of attributes.

Example:- here we have a student relation given:-

Roll_no.	Name	Email	Marks	Course_Id	Aadhar_no
1	A	ag@gm.com	68	011	5734
2	B	ab@gm.com	70	011	4567
3	C	bc@gm.com	56	022	7890
4	A	ac@gm.com	68	022	3456
5	D	ad@gm.com	80	011	5679
6	E	be@gm.com	65	033	4386
7	B	bd@gm.com	85	044	9374

Types of Functional dependencies in DBMS

There are 4 types of functional dependencies:-

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.
If X → Y and Y is the subset of X, then it is called trivial functional dependency.
i.e. X→X or Y→Y or XY→Y or XY→X is a trivial FD.

Example:-

Roll_no.	Name	Email
1	A	ag@gm.com
2	B	ab@gm.com
3	C	bc@gm.com
4	A	ac@gm.com
5	D	ad@gm.com

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.
When X intersection Y is NULL ($X \cap Y = \phi$), then $X \rightarrow Y$ is called as complete non-trivial functional dependencies.
If $X \rightarrow Y$ and Y is strictly not a subset of X, then it is called Non-trivial functional dependency.

Example:-

Roll_no.	Name	Email
1	A	ag@gm.com
2	B	ab@gm.com
3	C	bc@gm.com
4	A	ac@gm.com
5	D	ad@gm.com

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} → email is also a non-trivial functional dependency, since email 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.

Example:-

Roll_no.	Name	Marks	Course_Id
1	A	68	011
2	B	70	011
3	C	70	022
4	A	68	022
5	D	80	011
6	E	65	033
7	B	85	044

Here, roll_no → {name, marks} is a multivalued functional dependency, since the dependents name & marks are not dependent on each other (i.e. name → marks or marks → 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

Example:-

Roll_no.	Name	Marks	Course_Id
1	A	68	011
2	B	85	011
3	C	70	022
4	A	68	022
5	D	80	011
6	E	65	033
7	B	85	044

Here, roll_no → name and name → marks, Hence, according to the axiom of transitivity, roll_no → marks is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Attribute closure/Closure Set of attribute

- Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.
- Closure of attribute set {X} is denoted as $\{X\}^+$.
- Closure of a set F of FDs is the set F^+ of all FDs that can be inferred from F.
- Closure of a set of attributes X concerning F is the set X^+ of all attributes that are functionally determined by X.

Example:-

Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-

A → BC, BC → DE, D → F, CF → G

Now, let us find the closure of some attributes and attribute sets-

Closure of attribute A:-

A^+ = { A }

= { A , B , C } (Using A → BC)

= { A , B , C , D , E } (Using BC → DE)

= { A , B , C , D , E , F } (Using D → F)

= { A , B , C , D , E , F , G } (Using CF → G)

Thus,

A^+ = { A , B , C , D , E , F , G }

Closure of attribute D-

D^+ = { D }

= { D , F } (Using D → F)

Finding the Keys Using Closure

1. **Super Key:-** If the closure result of an attribute set contains all the attributes of the relation, then that attribute set is called as a super key of that relation.

Thus, we can say-

"The closure of a super key is the entire relation schema."

Or "The attribute closure of a super key contains all the attributes of table."

Example:-

Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-

A → BC, BC → DE, D → F, CF → G

A^+ = { A , B , C , D , E , F , G }

The closure of attribute A is the entire relation schema.

Thus, attribute A is a super key for that relation.

2. **Candidate Key:-** If there exists no proper subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

Note:-If the right side of functional dependency contains any one attribute of the prime attributes then there are more than one candidate key exist.

Example1:-

A → BC, BC → DE, D → F, CF → G

A^+ = { A , B , C , D , E , F , G }

attribute A is a super key for that relation.

No proper subset of attribute A contains all the attributes of the relation.

Thus, attribute A is also a candidate key for that relation.

Example2:-

Consider a relation R (A , B , C , D , E) with the functional dependencies-

A → B, B → CD, AD → E

A^+ = {A,B,C,D,E}

B^+= {B,C,D}

C^+= {C}

AB^+= {A,B,C,D,E}

AD^+= {A,D,E,B,C}

AC^+= {A,C,B,D,E}

D^+= {D}

Here AB, AD and AC contains all the attributes of the relation, but the proper subset {A } of all the closure also contains all the attributes of the relation, therefore only A is a candidate key.

Normalization in DBMS

Normalization is the process of organizing the data and the attributes of a database. It is performed to reduce the data redundancy in a database and to enhance data integrity in the table.

Need of normalization

- If we store all the data in one relation (table) then it may result in data redundancy (duplication). Data redundancy in DBMS means having the same data but at multiple places.
- It is necessary to remove data redundancy because it causes anomalies in a database which makes it very hard for a database administrator to maintain it.
- In DBMS, anomaly means the inconsistency occurred in the relational table during the operations performed on the relational table.
- Due to database anomalies, the integrity of the database suffers.
- Removing the anomalies is the main reason for relation normalization.

  There are major 3 types of anomalies:
  1. Insertion anomalies
  2. Updation anomalies
  3. Deletion anomalies

1. Insertion Anomalies

- Cannot insert student data without branch information, leading to NULL values.
- Redundant branch information is repeated when inserting multiple students of the same branch.
- Results in disk space wastage and inefficient resource utilization.

2. Update Anomalies

- Changing the HOD or address requires updating multiple records.
- Missing updates can cause data inconsistency.
- Increases the likelihood of errors and inconsistencies.

3. Deletion Anomalies

- Deleting a student may also delete necessary branch information.
- Losing the only student in a branch can result in the loss of branch data.
- Leads to unintended data loss.

Types of Normal Form in SQL

- As we have learned Normalization is the process of minimizing redundancy from a relation or set of relations.
- Normal forms are used to eliminate or reduce redundancy in database tables.

There are following type of Normal form:-

1. 1NF (First Normal Form)
2. 2NF (Second Normal Form)
3. 3NF (Third Normal Form)
4. BCNF (Boyce-Codd Normal Form)
5. 4NF (Fourth Normal Form)
6. 5NF (Fifth Normal Form)

1. 1NF (First Normal Form)

- In 1NF we depend on ER diagram if ER diagram is designed error free and clear then the relations (tables) are also contain atomic value.
- In first normal form data redundancy may increases as the same data is in multiple rows to ensure the atomicity.

- A relation is in first normal form if:
  - It does not contain any composite or multi-valued attribute or it should only have single (atomic) valued attributes/columns.
  - Values stored in a column should be of the same domain. For example: In name(char) attribute we store only name not phone number(int).
  - All the columns in a table should have unique names.
  - The order in which data is stored, does not matter.

roll_no	name	subject	address
101	Raj	OS, CN	Amritsar, Punjab, India
103	Vijay	Java	Gurugram, Haryana, India
102	Sohan	C, C++	Prayagraj, UP,India

Example:- Every row contains single (atomic) value which may increase the data redundancy.

roll_no	name	subject	city	state	country
101	**Raj**	**OS**	**Amritsar**	**Punjab**	**India**
101	**Raj**	**CN**	**Amritsar**	**Punjab**	**India**
103	**Vijay**	**Java**	**Gurugram**	**Haryana**	**India**
102	**Sohan**	**C**	**Prayagraj**	**UP**	**India**
102	**Sohan**	**C++**	**Prayagraj**	**UP**	**India**

1. **We can create another table for multivalued attribute subject.**

roll_no	name	city	state	country
101	Raj	Amritsar	Punjab	India
103	Vijay	Gurugram	Haryana	India
102	Sohan	Prayagraj	UP	India

roll_no	subject
101	OS
101	CN
103	Java
102	C
102	C++

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

2. 2NF (Second Normal Form)

- If a relation is in Second Normal Form, it must satisfy two conditions:
  - The table should be in the First Normal Form.
  - There should be no Partial Dependency.
  - In other words non prime attributes should not be determined by proper subset of candidate key (prime attributes)
- If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible (no proper subset of candidate key exist)).
- If all the attributes of a relation is prime attributes then that relation is always in 2NF.

Partial Dependency:- If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

i.e. if X → A holds, then there should be any proper subset Y of X, for which Y → A also holds true.

i.e. proper subset of candidate key→Non prime attribute

Example1 for 2NF:-

Consider following functional dependencies in relation R (A, B , C, D)

{AB -> C , BC -> D}

Solution:- find the closure for candidate key

AB^+={A,B,C,D}

BC^+={B,C,D}

Prime attributes={A,B} (attributes which participates in candidate key)

Non- prime attributes={C,D} (attributes other than candidate key)

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute. So this relation is in 2NF.

Example2:-Given a relation R(A,B,C,D,E) and Functional Dependency set FD = { AB → C, D → E }, determine whether the given R is in 2NF or not.

Solution:- Here first of all we have to find candidate key so we will find the attribute closure of the given attributes. Attribute whose closure contains all the attributes and its proper subset is not a super key would be a candidate key.

AB^+={A,B,C}

ABC^+={A,B,C}

ABD^+= {A,B,C,D,E}

BC^+={B,C}

Prime attributes={A,B,D} (attributes which participates in candidate key)
Non- prime attributes={C,E} (attributes other than candidate key)
In the above relation, ABD is the only candidate key and partial prime attribute (AB) determine non prime attribute (C). Also partial prime attribute (D) determine non prime attribute (E).
therefore this relation is not in 2NF.

3. 3NF (Third Normal Form)

- A relation is in third normal form if:
  a) It is in second normal form.
  b) No non-prime attribute is transitively dependent on the candidate (primary) key. (i.e. NPA → NPA)
  In other words:-
- A relation is in 3NF if at least one of the following condition holds in every non-trivial functional dependency X –> Y
  a) Either X is a super key.
  b) Or Y is a prime attribute (each element of Y is part of some candidate key).
- 3NF is used to reduce the data duplication.
- It is also the used to achieve data integrity.
- If all attributes of relation are prime attribute, then the relation is always in 3NF.

Example1 for 3NF:- Consider relation R(A, B, C, D, E) with functional dependencies {A -> BC, CD -> E, B -> D, E -> A}
Solution:- First of all find the candidate keys
A^+= { A,B,C,D,E}
E^+= {A,B,C,D,E}
CD^+={C,D,E,A,B}
BC^+= {B,C,D,E,A}
Candidate keys:-{A, E, CD, BC}
Prime attributes: {A,E,C,D,B}
There is no non prime attribute hence there is no partial dependency and this relation is in 2NF.
Also there is no case of NPA→NPA.
Therefore this relation is in 3NF.

Example2 for 3NF:- Consider relation R(A, B, C, D, E) with functional dependencies {A→B, A→C, C→D, A→E}
Solution:- First of all find the candidate keys
A^+= { A,B,C,D,E}
Candidate keys:-{A}
Prime attributes: {A}
non prime attributes: {B,C,D,E}
Now check the partial dependency i.e. proper subset of candidate key determines non prime attribute. In this case candidate key has one attribute it means only one prime attribute. Hence no case of partial dependency.
So, this relation is in 2NF.
Now check if non prime attribute determines non prime attribute.
C→D both C & D are non prime attributes. Hence this relation is not in 3NF.

Example3 for 3NF:- Consider following functional dependencies in relation R (A, B , C, D) {AB -> C,BC -> D}

Solution:- First of all find the candidate keys

AB^+= { A,B,C,D}

Candidate keys:-{AB}

Prime attributes: {A,B}

non prime attributes: {C,D}

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute. This is in 2NF.

Now for 3NF check either LHS is superkey or RHS is the prime attribute.

i.e. AB→C here AB is a super key no need to check RHS.

BC→D here BC is not a super key now check RHS. RHS is non prime attribute. Therefore this relation is not in 3NF.

Here we can say that this relation is not in 3NF.

4. Boyce-Codd Normal Form (BCNF)

- BCNF is an extension to the third normal form, and is also known as 3.5 Normal Form.
- A relation R is in BCNF if
  - R is in Third Normal Form
  - For every non-trivial functional dependency X –> Y, X (LHS) is a super key.
- When a database has composite candidate key then sometimes in 3NF it creates some anomaly. BCNF is used to reduce these anomalies. BCNF is best normal form for relation.
- Every Binary Relation (a Relation with only 2 attributes) is always in BCNF
- BCNF is not always dependency preserving.

Example1:- A relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}

Solution:- First of all find that the LHS of all the functional dependency is super key or not. If LHS is not a super key it means this relation is not in BCNF.

BC^+={B,C,D,E}

AC^+= { A,B,C,D,E}

B^+= {B,E}

Here BC and B is not a super key.

Therefore this relation is not in BCNF.

Now check if it is in 3NF or not

Candidate keys: {AC}

Prime attributes: {A,C}

non prime attributes: {B,D,E}

Now check if non prime attribute determines non prime attribute.

i.e. BC→D and B→E (NPA→NPA)

Therefore this relation is not in 3NF.

For 2NF check if proper subset of candidate key determines non prime attribute.

Here no partial dependency exist

Therefore this relation is in 2NF.

Example2:- Consider relation R(A, B, C) with FD set {A -> B, B -> C, C->A}

Solution:- First of all find that the LHS of all the functional dependency is super key or not. If LHS is not a super key it means this relation is not in BCNF.

A^+= {A,B,C}

B^+= {B,C,A}

C^+= {C,A,B}

Here A,B,C all are super key.

Therefore this relation is in BCNF.

Some important points about normal form

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also satisfied and all the lower normal forms (2NF,1NF) are satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Advantages of Normalization

- Utilizing normalized database or data redundancy
- Duplication might be removed.
- Reduced null values may result from normalization.
- results in a more compact database (since there is less data duplication or zero).
- Reduce/avoid problems caused by data alteration.
- The database's structure is clearer and easier to understand.
- The database can have existing data added to it without any negative effects.

Disadvantages of Normalization

- Connecting multiple tables makes the database more complex and harder to manage.
- Data is stored as codes instead of real information, needing frequent lookups.
- Writing queries is harder due to complicated SQL and system design.
- The database runs slower than simpler structures.
- Proper knowledge is needed for normalization; mistakes can lead to errors and data issues.

Relational Decomposition

- Relational decomposition refers to breaking a large or complex relation (table) into smaller, simpler relations without losing important data.
- This process helps in eliminating redundancy and anomalies while maintaining the integrity of the data.

Types of Decomposition

1. **Lossless Decomposition:**
   - Ensures that no data is lost when the decomposed tables are joined back to form the original relation.
   - Condition: If a relation R is decomposed into R1 and R2, the join R1⋈R2 should produce the original relation R.
   - **Example:**
     Original relation R(Student,Course,Instructor) is decomposed into:
     R1(Student,Course)
     R2(Course,Instructor)
     The join of R1 and R2 will give back the original relation.

2. **Lossy Decomposition:**
   - When some information is lost, or additional meaningless tuples are generated during the join of decomposed relations.
   - It is not desirable as it leads to incorrect results.
   - Example:
     If decomposed tables create duplicates or null values after joining, it results in a lossy decomposition.

Properties of Decomposition

1. Dependency-Preserving Decomposition:

- Ensures that all functional dependencies (FDs) in the original relation are preserved in the decomposed relations.
- This property helps in maintaining data integrity without requiring complex joins to enforce FDs.
- **Example:**
  If a relation R has FDs:
  A→B, B→C
  The decomposition should preserve these dependencies in the resulting tables.

2. Lossless-Join Decomposition:

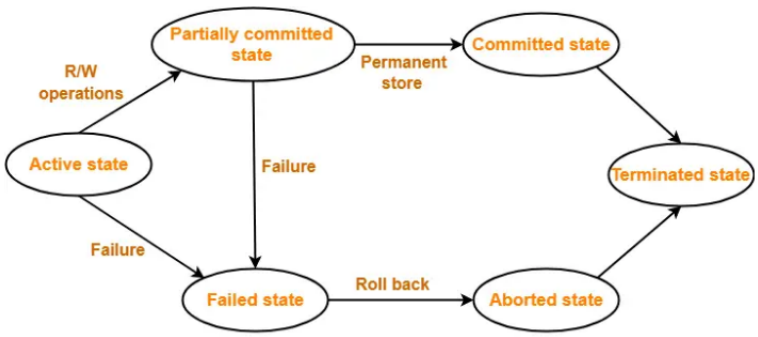
- Ensures that no data is lost when the decomposed tables are joined back.
- Condition: At least one common attribute between decomposed tables must act as a key in one of the decomposed tables.

Example:
If relation R is decomposed into R1 and R2, the common attribute should be a key in either R1 or R2.

Transactions

- A transaction is a sequence of one or more database operations (such as insert, delete, update, or read) executed as a single unit of work.
- The primary purpose of a transaction is to ensure data consistency and integrity, even in the presence of failures or concurrent access.

States of Transactions



State Transition Diagram for a Database Transaction

The various states of a transaction concept in DBMS are listed below:

State	Transaction types
Active State	This is the initial state of a transaction when it starts its execution. During this state read or write operations can be performed. After completion of all the read and write operation successfully the transaction enters the partially committed state , if somehow any operation fails, then it enters to failed state.
Partially Committed	If transaction enters in partially committed state it means the read and write operation is successfully completed. In this state, the changes which were previously made in the main memory are now made permanent in the database, after which the state will progress to committed state but in case of a failure it will go to the failed state
Committed State	After the transaction successfully passes the partially committed state, it moves to the committed state. In this state, the changes made by the transaction are permanently saved to the database, and the transaction is considered complete.
Failed State	If any operation during the transaction fails due to some software, hardware or violates any integrity constraints (e.g., foreign key constraints, unique constraints), then it goes to the failed state. In this state, the changes made by the transaction are typically rolled back, and the database is left unchanged.
Aborted State	If the transaction fails during its execution, it goes from failed state to aborted state and because in the previous states all the changes were only made in the main memory, these uncommitted changes are either deleted or rolled back. The transaction at this point can restart and start afresh from the active state.
Terminated State	Once a transaction reaches either the committed or aborted state, it has reached its final conclusion and is no longer actively processing or affecting the database and the transaction is terminated.

ACID Properties of Transactions

The ACID properties ensure that database transactions are processed reliably.

1. **Atomicity**

- A transaction must be treated as an atomic unit, meaning either all operations within the transaction are completed, or none are.
- Log files or journals are used to keep track of operations to ensure rollbacks when necessary.
- Example: If a bank transfer fails after debiting the source account but before crediting the destination account, the entire transaction should be rolled back.

2. **Consistency**

A transaction must maintain the integrity constraints of the database. After the transaction, the database should remain in a consistent state.
The system ensures data integrity by enforcing rules and constraints during transactions.
Example: Inserting a record in a table that violates a foreign key constraint should not be allowed.

3. **Isolation**

- Transactions should execute independently without interference from other concurrent transactions.
- Levels of Isolation:
- Read Uncommitted: Transactions can see uncommitted changes from other transactions.
- Read Committed: Transactions can only see committed changes.
- Repeatable Read: Ensures the same data is read multiple times within a transaction.
- Serializable: The highest level, ensuring complete isolation.
- Example: If two users are booking the last seat on a flight, the system should isolate these transactions to prevent double-booking.

4. **Durability**

- Once a transaction is committed, its changes must persist, even in the case of a system failure.
- Data is written to non-volatile storage like disks or backups.
- Example: After successfully booking a ticket, the booking details should remain saved even if the server crashes.

Importance of ACID Properties in Transactions:

- Data Integrity: Ensures that data remains accurate and consistent.
- Fault Tolerance: Prevents data loss in case of crashes or failures.
- Concurrency Control: Allows multiple transactions to run concurrently without conflicts.
- User Trust: Provides a reliable system for users and applications.

Important Point

Property	Responsibility for maintaining properties
Atomicity	Transaction Manager
Consistency	Application programmer
Isolation	Concurrency Control Manager
Durability	Recovery Manager

Concurrency Control

- Concurrency control ensures that multiple transactions can be executed simultaneously without compromising the consistency and integrity of the database.
- It manages conflicts that arise when transactions access shared data concurrently.

Issues in Concurrency Control

A. **Lost Update:** Occurs when two transactions read the same data and then update it, but the final update from one transaction overwrites the update from the other, leading to data loss.

Example:

- Transaction T1 reads a balance of ₹10,000.
- Transaction T2 also reads the same balance.
- T1 adds ₹500 and updates the balance to ₹10,500.
- T2 adds ₹1,000 and updates the balance to ₹11,000, overwriting T1's update.

Solution: Use locking mechanisms or serialization to prevent concurrent updates.

B. **Dirty Read:-** Occurs when a transaction reads uncommitted data from another transaction. If the other transaction rolls back, the data read becomes invalid.

Example:

- Transaction T1 updates a product price from ₹100 to ₹120 but hasn't committed.
- Transaction T2 reads the new price of ₹120.
- T1 rolls back, but T2 has already used the incorrect price.

Solution: Apply stricter isolation levels like Read Committed or above.

C. **Non-Repeatable Read:-** Occurs when a transaction reads the same data twice and finds different values due to another transaction modifying the data between the reads.

Example:

- Transaction T1 reads a balance of ₹5,000.
- Transaction T2 updates the balance to ₹6,000 and commits.
- T1 reads the balance again and finds ₹6,000, leading to inconsistency.

Solution: Use isolation levels like Repeatable Read to prevent changes between reads.

Read write conflict

A read-write (RW) conflict occurs when a transaction reads data, and another transaction writes the same data before the read is complete.

1. **Read – Read (RR):** Two transactions trying to read the same data item at the same time.
2. **Write-Write (WW):** Two transactions trying to write the same data item at the same time.
3. **Write-Read (WR):** One transaction writes and another reads the same data item before the write is committed.
4. **Read-Write (RW):** One transaction reads data and another writes the same data item before the read is completed.

Read – Read: No conflict.
Read- Write: Conflict
Write- Read: Conflict
Write- Write: Conflict

Schedules

Schedules:- A schedule is the sequence in which operations from multiple transactions are executed.

Types of Schedules

1. **Serializable Schedule:** A schedule is said to be serializable if its outcome is equivalent to some serial execution of the transactions.It ensures consistency by executing transactions in a logically isolated manner.

2. **Non-Serializable Schedule(parallel):** A schedule that does not guarantee a consistent state and may lead to anomalies.

Serializability

- Serializability is a concept in database concurrency control that ensures the results of executing transactions concurrently are the same as if they were executed serially, one after another.
- It guarantees database consistency and prevents anomalies caused by concurrent transactions.
- A schedule is serializable if it produces the same result as some serial execution of the same transactions.

Types of Serializability

1. **Conflict Serializability:** A schedule is conflict-serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. A directed Precedence Graph (Dependency Graph) used to determine if a schedule is conflict-serializable.

Conflicting Operations:

Two operations are said to be in conflict if:

- They are from different transactions.
- They access the same data item.
- At least one of them is a write operation.

Example:

Schedule:
T1: Read(A), Write(A)
T2: Read(A), Write(A)
Conflicts between T1's Write(A) and T2's Read(A).

2. **View Serializability:** A schedule is view-serializable if it results in the same final state as a serial schedule when viewed in terms of the data read and written by transactions.

Components of View Equivalence:

Initial Read: Each transaction reads the same initial value.
Final Write: Each transaction writes the same final value.
Intermediate Read: If a transaction reads a value written by another transaction in the schedule, the same should happen in the serial schedule.
More Flexible: View serializability allows schedules that are not conflict-serializable but still ensure correct results.

Importance of Serializability

- **Data Consistency:** Ensures that concurrent transactions do not violate integrity constraints.
- **Concurrency Control:** Balances the need for performance with the need for correctness.
- **Isolation:** Provides the highest level of isolation in databases, preventing anomalies like Lost Updates and Dirty Reads.

Anomalies Prevented by Serializability

- **Lost Update:** Ensures no update is overwritten by another transaction.
- **Dirty Read:** Prevents reading uncommitted changes from other transactions.
- **Non-Repeatable Read:** Ensures consistent reads within a transaction.
- **Phantom Reads:** Prevents new records from appearing in subsequent reads.

Techniques for Concurrency Control

- **Locks:** Mechanisms to restrict access to data during transactions.
- **Timestamp Ordering:** Assigns timestamps to transactions to maintain order.
- **Optimistic Concurrency Control:** Assumes minimal conflict and checks for conflicts at commit time.
- **Multiversion Concurrency Control (MVCC):** Maintains multiple versions of data for read consistency.

Locking Protocols

- Locking protocols are mechanisms used in databases to control access to data during concurrent transactions.
- They ensure data consistency and prevent conflicts by allowing only one transaction to access critical data at a time, depending on the type of lock.

Shared vs. Exclusive Locks

1. **Shared Lock (S-lock):** Allows multiple transactions to read a data item but prevents any transaction from modifying it.

   Used for read-only operations where data integrity needs to be maintained.
   Multiple transactions can acquire a shared lock on the same data item.
   No transaction can write to the data while shared locks are held.
   Example: Transaction T1 and T2 can both read a balance simultaneously if they hold shared locks on it.

2. **Exclusive Lock (X-lock):** Allows a single transaction to both read and modify a data item. No other transaction can access the data until the lock is released.

   Used for update operations to maintain data consistency.
   Only one transaction can hold an exclusive lock on a data item.
   Prevents both read and write access by other transactions.
   Example: If T1 holds an exclusive lock on a balance, no other transaction can read or write to it until T1 releases the lock.

Two-Phase Locking Protocol (2PL)

The Two-Phase Locking Protocol ensures serializability by dividing the locking process into two distinct phases:

Phases of 2PL:

Growing Phase:

- A transaction can acquire locks (both shared and exclusive) but cannot release any locks.
- This phase ensures that the transaction locks all necessary data items before making any changes.

Shrinking Phase:

- A transaction can release locks but cannot acquire any new locks.
- Once a transaction starts releasing locks, it cannot lock any new data items.

Types of 2PL

Strict Two-Phase Locking (Strict 2PL):

- All exclusive locks are held until the transaction commits or aborts.
- Prevents cascading rollbacks by ensuring no other transaction reads uncommitted data.

Rigorous Two-Phase Locking:

- Both shared and exclusive locks are held until the transaction completes, providing even stricter control than strict 2PL.

Example of Two-Phase Locking:

Transaction T1 needs to read and update two data items (A and B):

Growing Phase:

- T1 acquires a shared lock on A and reads it.
- T1 upgrades to an exclusive lock on A to update it.
- T1 acquires an exclusive lock on B to update it.

Shrinking Phase:

- T1 releases the locks on A and B after completing the updates.

Advantages of Two-Phase Locking

- Guarantees serializability of transactions.
- Ensures data consistency and integrity.
- Prevents anomalies like Lost Updates and Dirty Reads.

Disadvantages of Two-Phase Locking

- May lead to deadlocks if two or more transactions wait indefinitely for each other's locks.
- Can reduce concurrency and cause delays.

Deadlocks in Two-Phase Locking

A deadlock occurs when two or more transactions are waiting for each other to release locks, leading to an infinite wait.

Example:

- Transaction T1 locks A and waits for B.
- Transaction T2 locks B and waits for A.

Solutions to Deadlocks:

- Deadlock Prevention: Allocate resources in a specific order.
- Deadlock Detection and Recovery: Use a wait-for graph to detect and terminate one transaction to break the cycle.
- Timeout: Abort a transaction if it waits too long.