

# DBMS Mid 1 QnA

## UNIT - 1

### 1. What is DBMS? Compare and Contrast file Systems with database systems.

**Database Management System (DBMS)** is a software for storing and retrieving user's data while considering appropriate security measures. It consists of a group of programs that manipulate the database. The DBMS accepts the request for data from an application and instructs the DBMS engine to provide the specific data. In large systems, a DBMS helps users and other third- party software to store and retrieve data.

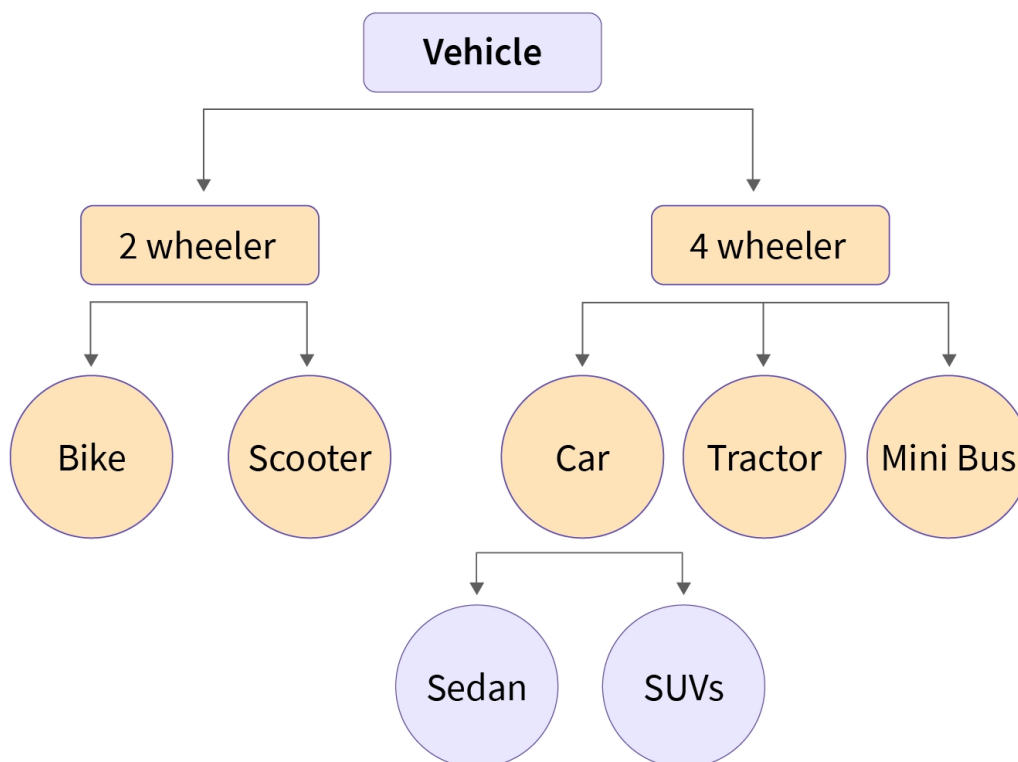
	FILE SYSTEM	DBMS
Description	Used to manage and organise the files stored in the hard disk of the computer	A software to store and retrieve the user's data
Redundancy	Redundant data is present	No presence of redundant data
Query Processing	Query processing is efficient	Query processing is not efficient
Data Consistency	Data consistency is low	Due to the process of normalization, the data consistency is high
Complexity	Less complex, does not support complicated transactions	More complexity in managing the data, easier to implement complicated transactions
Security	Less security	Supports more security mechanisms
Cost	Less expensive in comparison to DBMS	Higher cost than the File system
Crash Recovery	Does not support crash recovery	Crash recovery mechanism is highly supported

### 2. Discuss different types of Data models.

Data models in DBMS help to understand the design at the conceptual, physical, and logical levels as it provides a clear picture of the data making it easier for developers to create a physical database. Data models are used to describe how the data is stored, accessed, and updated in a DBMS. A set of symbols and text is used to represent them so that all the members of an organization can understand how the data is organized. It provides a set of conceptual tools that are vastly used to represent the description of data. There are many types of data models that are used in the industry.

## 1. Hierarchical Model

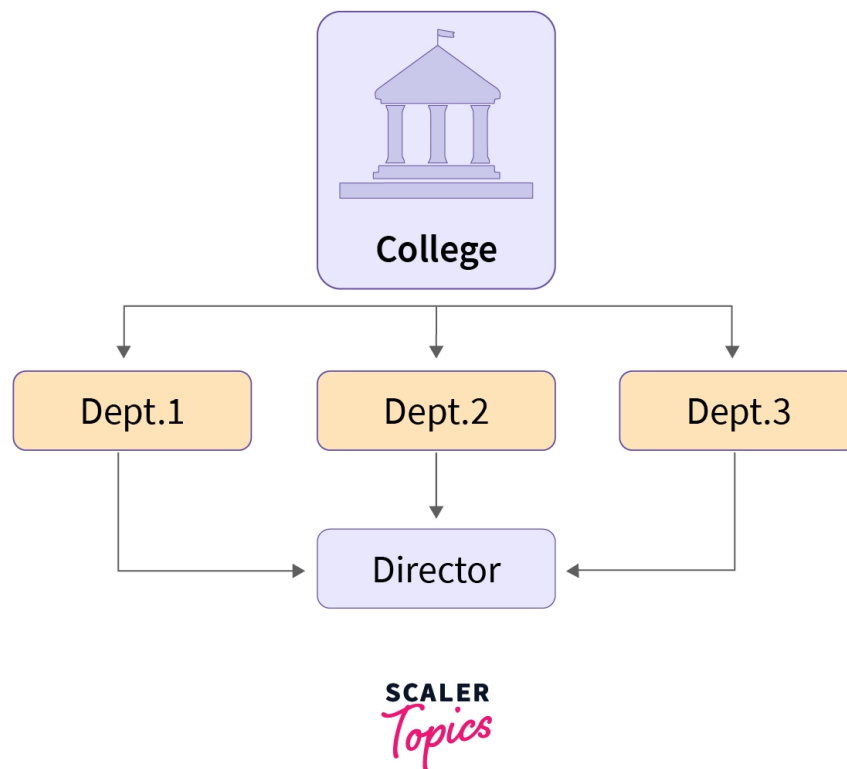
- This database model organizes data into a tree-like structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.
- In this model, a child node will only have a single parent node. This model efficiently describes many real-world relationships like index of a book, recipes etc. In hierarchical model, data is organized into a tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of course many students.



SCALER  
Topics

## 2. Network Model

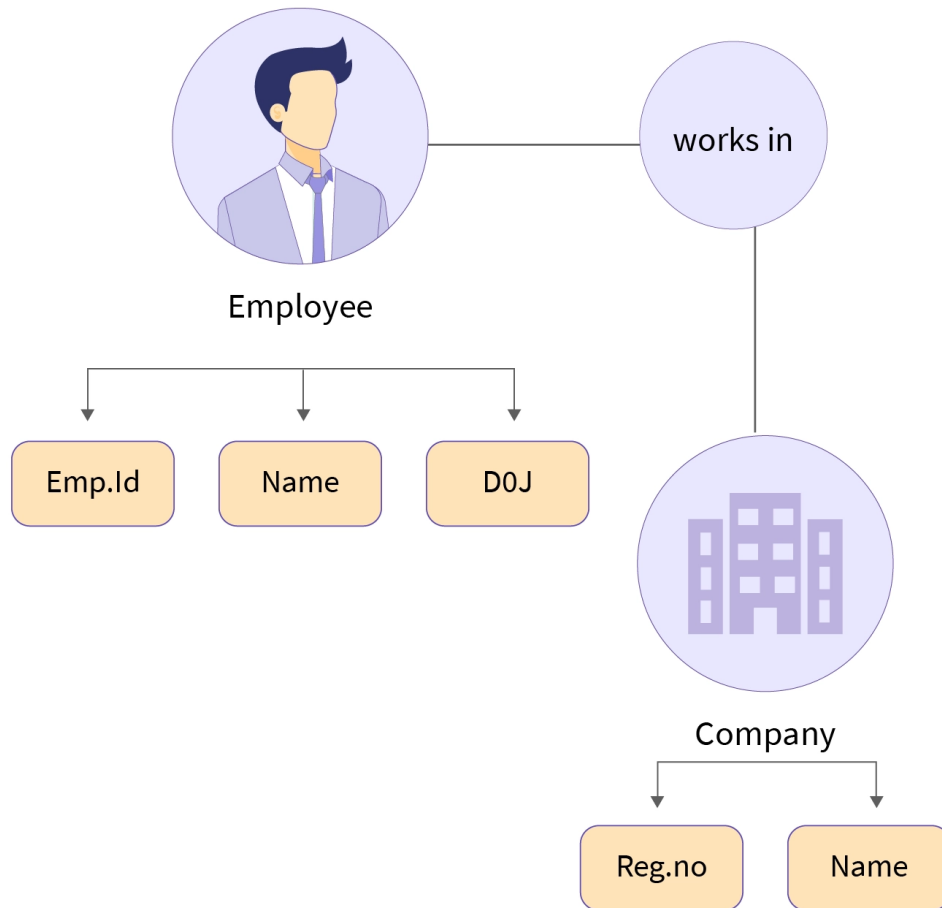
- This is an extension of the Hierarchical model. In this model data is organized more like a graph, and are allowed to have more than one parent node.
- In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships. This was the most widely used database model, before Relational Model was introduced.



### 3. Entity-relationship Model

- In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.
- Different entities are related using relationships. E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.
- Let's take an example, If we have to design a School Database, then Student will be an entity with attributes name, age, address etc. As Address is generally complex, it can be another entity with attributes street name, pincode, city etc, and there will be a

relationship between them.



SCALER  
Topics

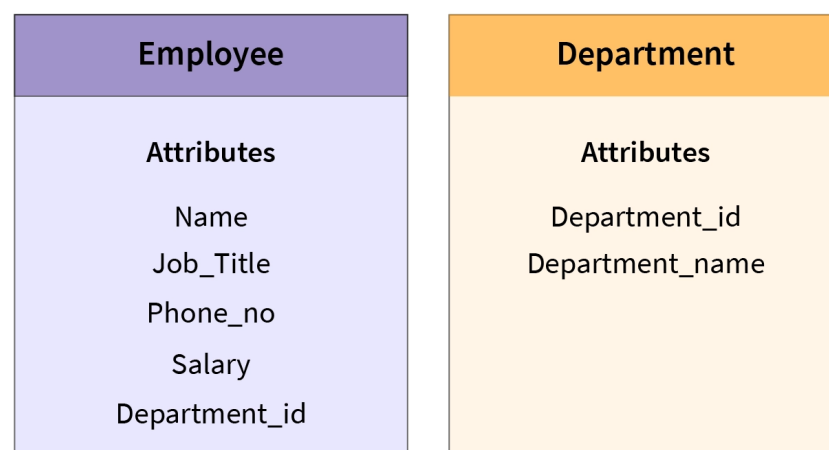
#### 4. Relational Model

- In this model, data is organized in two-dimensional tables and the relationship is maintained by storing a common field.
- This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, in fact, we can say the only database model used around the world.
- The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table. Hence, tables are also known as relations in relational model.

Stu. Id	Name	Branch
101	Naman	CSE
102	Saloni	ECE
103	Rishabh	IT
104	Pulkit	ME

### 1. Object-Oriented Data Model

- As suggested by its name, the object-oriented data model is a combination of object-oriented programming and relational data model. In this data model, the data and their relationship are represented in a single structure which is known as an object.
- Since data is stored as objects we can easily store audio, video, images, etc in the database which was very difficult and inconvenient to do in the relational model. As shown in the image below two objects are connected with each other through links.



SCALER  
Topics

### 3. Explain the structure of DBMS with a neat diagram.

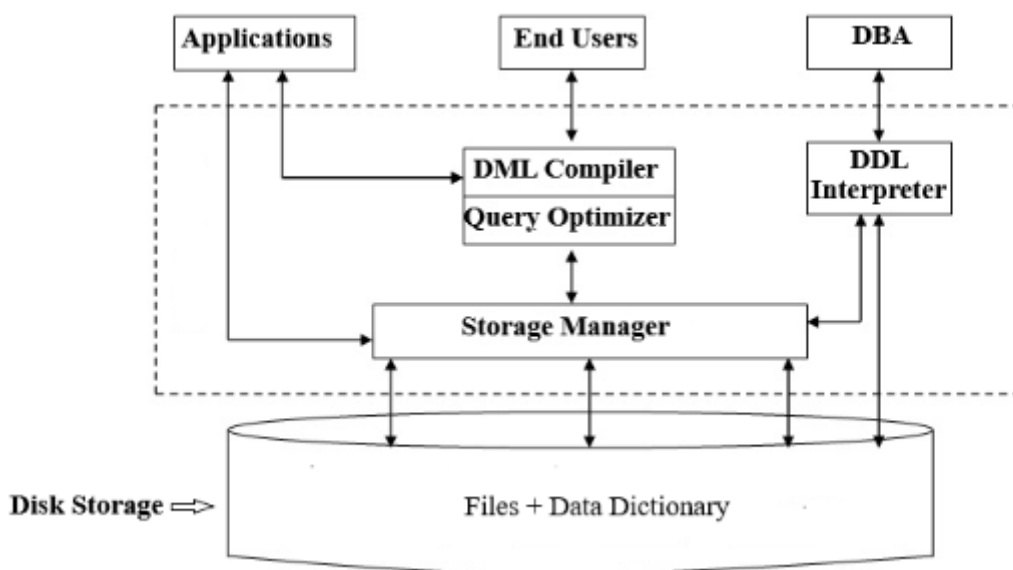
#### Structure of DBMS

DBMS (Database Management System) acts as an interface between the user and the database. The user requests the DBMS to perform various operations (insert, delete, update and retrieval) on the database. The components of DBMS perform these requested operations on the database and provide necessary data to the users. Database Management System (DBMS) is software that allows access to data stored in a database and provides an easy and effective method of:

- Defining the information.
- Storing the information.
- Manipulating the information.
- Protecting the information from system crashes or data theft.
- Differentiating access permissions for different users.

## Components of DBMS Structure

The structure of DBMS is divided into three components namely "Query Processor", "Storage Manager", and "Disk Storage". Each of these components plays a very crucial role in Database Management.



### 1. Query Processor

The query processing is handled by the query processor, as the name implies. It executes the user's query, to put it simply. In this way, the query processor aids the database system in making data access simple and easy. The query processor's primary duty is to successfully execute the query. The Query Processor transforms (or interprets) the user's application program-provided requests into instructions that a computer can understand.

### Components of the Query Processor

- **DDL Interpreter:** Interprets Data Definition Language (DDL) statements like those used in schema definitions (such as create, remove, etc.). This interpretation

yields a set of tables that include the meta-data (data of data) that is kept in the data dictionary.

- **DML Compiler:** Converts Data Manipulation Language (DML) statements like select, update, and delete into low-level instructions or machine-readable object code, and optimizes queries. Query optimization determines the most effective technique to carry out a query.
- **Embedded DML Pre-compiler:** Converts embedded DML commands in the application program into standard procedural calls.
- **Query Optimizer:** Analyzes and optimizes queries for efficient execution.

## 2. Storage Manager

An application called Storage Manager acts as a conduit between the queries made and the data kept in the database. It keeps the database's consistency and integrity by applying restrictions and running the Data Control Language (DCL) instructions. It is responsible for retrieving, storing, updating, and removing data from the database.

### Components of Storage Manager

- **Integrity Manager:** Manages integrity constraints when there is any change in the database.
- **Authorization Manager:** Verifies the user's validity and authentication for specific queries or requests.
- **File Manager:** Manages all the files and data structures of the database.
- **Transaction Manager:** Ensures database consistency before and after transactions and controls concurrent processes.
- **Buffer Manager:** Manages data transfer between primary and main memory and handles cache memory.

## 3. Disk Storage

A DBMS can use various kinds of Data Structures as a part of the physical system implementation in the form of disk storage.

### Components of Disk Storage

- **Data Dictionary:** Contains the metadata (data of data) detailing the structure of each object in the database.
- **Data Files:** Stores the data in files.
- **Indices:** Used to access and retrieve data efficiently.

## 4. What is the E-R model? Explain in detail about components of the E-R Model. Draw an E-R Diagram for Student Database

### Entity Relationship (ER) Diagram




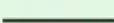

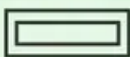
ER Diagram stands for Entity Relationship Diagram, also known as ERD, is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes, and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes, and diamond shapes to represent relationships.

## Symbols Used in ER Model

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

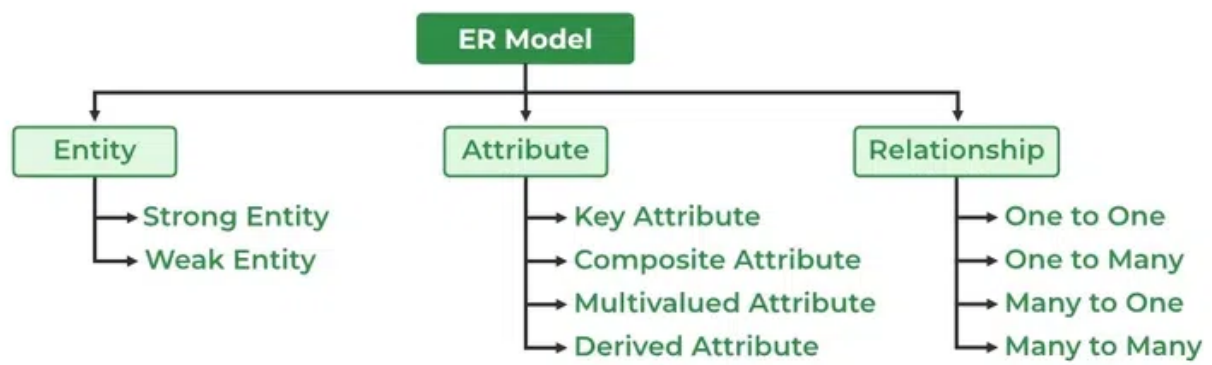
- **Rectangles**: Represent Entities in ER Model.
- **Ellipses**: Represent Attributes in ER Model.
- **Diamond**: Represent Relationships among Entities.
- **Lines**: Represent attributes to entities and entity sets with other relationship types.
- **Double Ellipse**: Represent Multi-Valued Attributes.
- **Double Rectangle**: Represent a Weak Entity.

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

## Components of ER Diagram

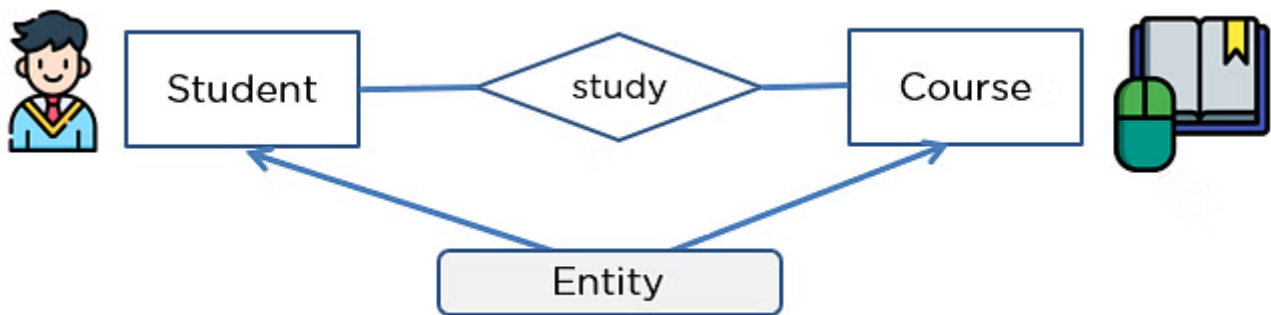
ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.





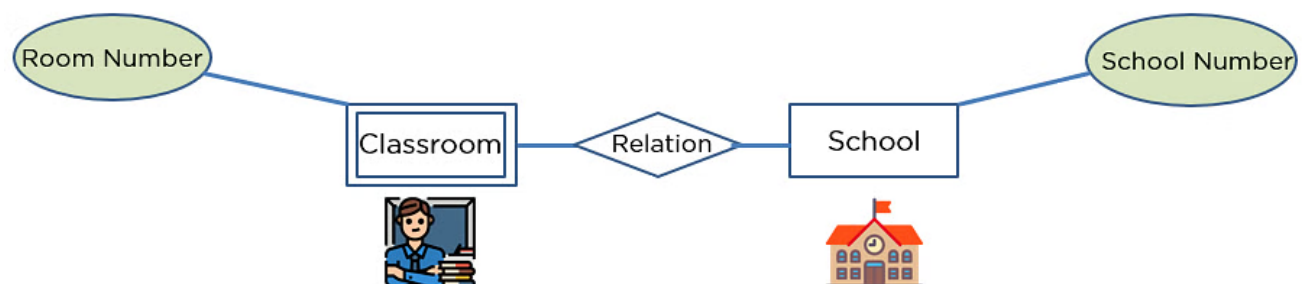
## Entities

An entity can be either a living or non-living component. It showcases an entity as a rectangle in an ER diagram. For example, in a student study course, both the student and the course are entities.



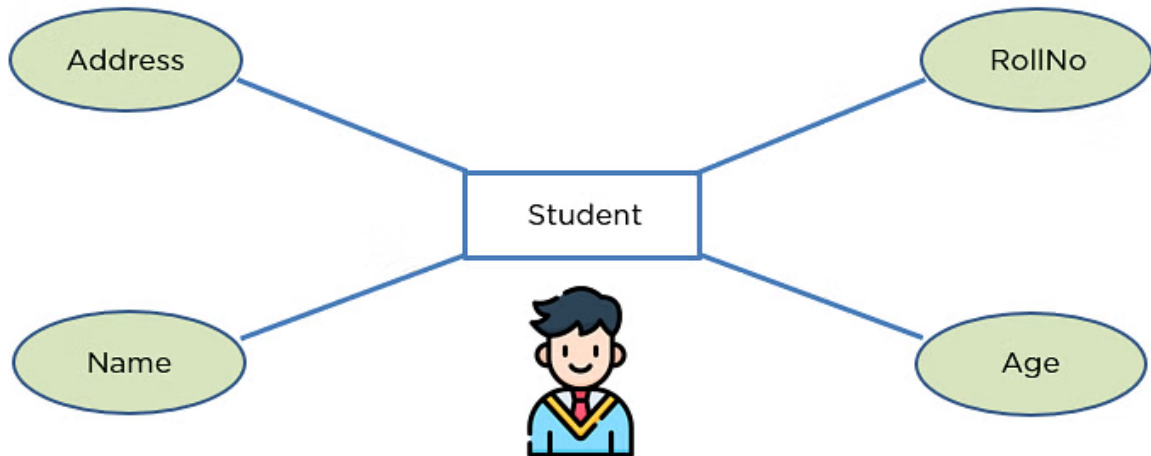
## Weak Entity

An entity that makes reliance over another entity is called a weak entity. You showcase the weak entity as a double rectangle in ER Diagram. In the example below, school is a strong entity because it has a primary key attribute - school number. Unlike school, the classroom is a weak entity because it does not have any primary key and the room number here acts only as a discriminator.



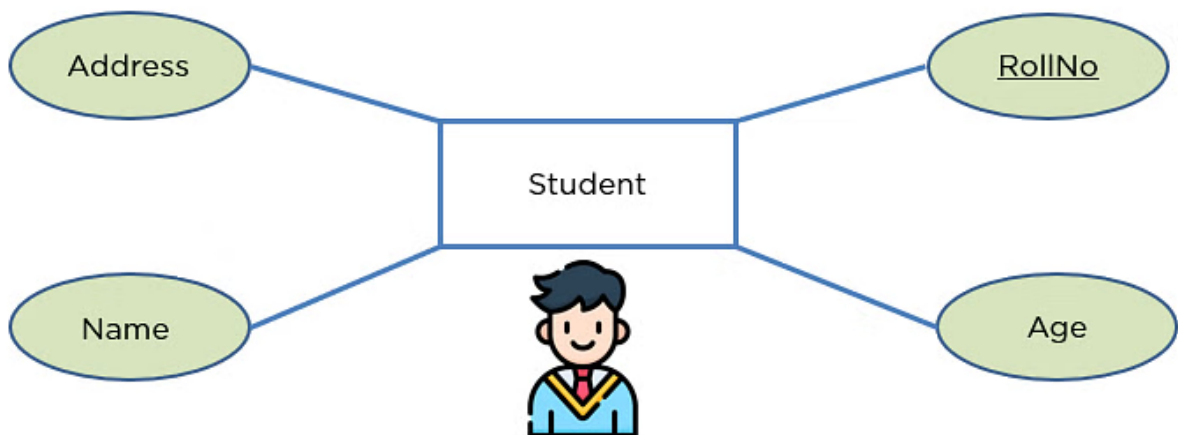
## Attribute

An attribute exhibits the properties of an entity. You can illustrate an attribute with an oval shape in an ER diagram.



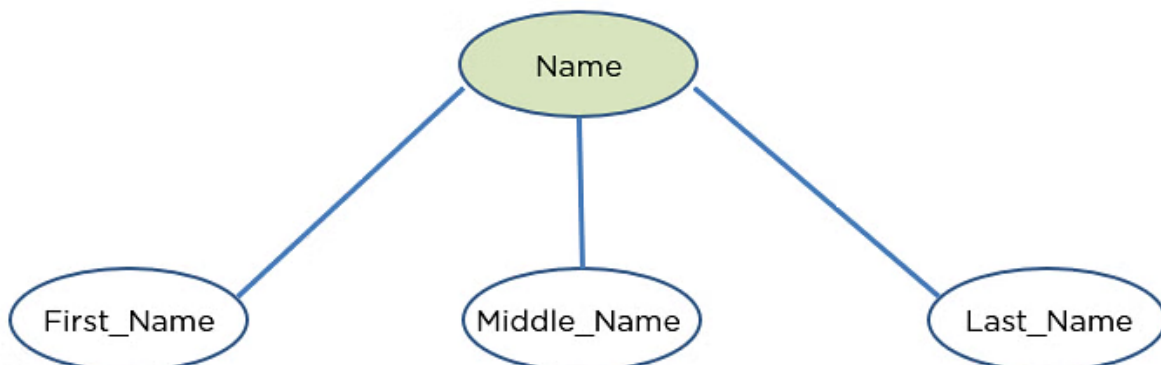
## Key Attribute

Key attribute uniquely identifies an entity from an entity set. It underlines the text of a key attribute. For example: For a student entity, the roll number can uniquely identify a student from a set of students.



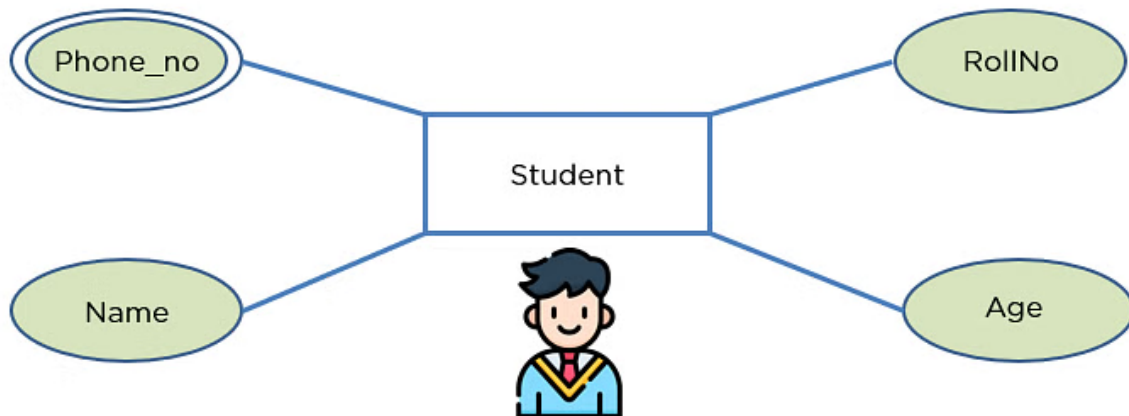
## Composite Attribute

An attribute that is composed of several other attributes is known as a composite attribute. An oval showcases the composite attribute, and the composite attribute oval is further connected with other ovals.



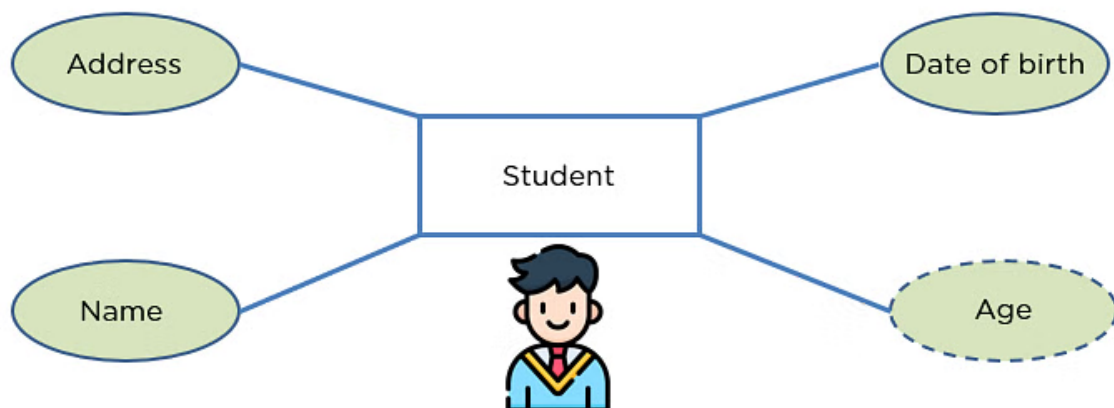
## Multivalued Attribute

Some attributes can possess over one value, those attributes are called multivalued attributes. The double oval shape is used to represent a multivalued attribute.



## Derived Attribute

An attribute that can be derived from other attributes of the entity is known as a derived attribute. In the ER diagram, the dashed oval represents the derived attribute.

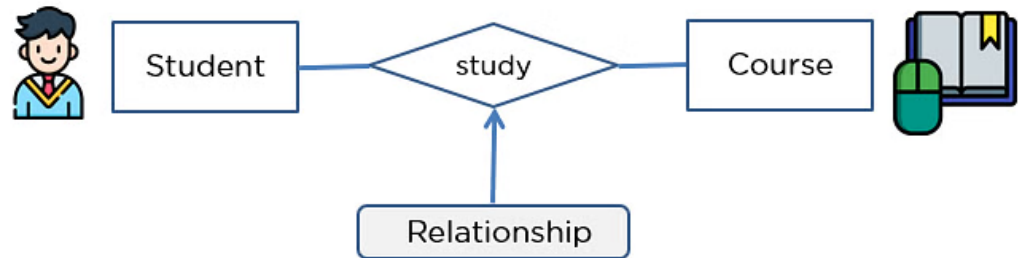


## Relationship

The diamond shape showcases a relationship in the ER diagram. It depicts the relationship between two entities. In the example below, both the student and the course are entities, and study is the relationship between them.

- **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.



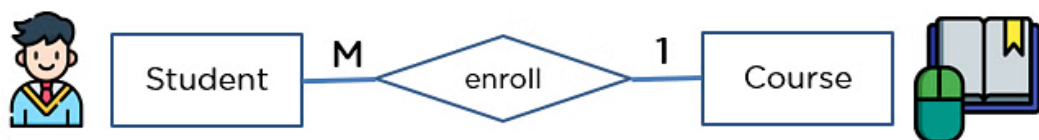
### • One-to-Many Relationship

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



### • Many-to-One Relationship

- When more than one element of an entity is related to a single element of another entity, then it is called a many-to-one relationship.
- For example, students have to opt for a single course, but a course can have many students.



### • Many-to-Many Relationship

- When more than one element of an entity is associated with more than one element of another entity, this is called a many-to-many relationship.
- For example, you can assign an employee to many projects and a project can have many employees.



## Participation Constraint

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

1. **Total Participation:** Each entity in the entity set must participate in the relationship.
2. **Partial Participation:** The entity in the entity set may or may NOT participate in the relationship.

## 5. What is data independence and how does a DBMS support it? Explain.

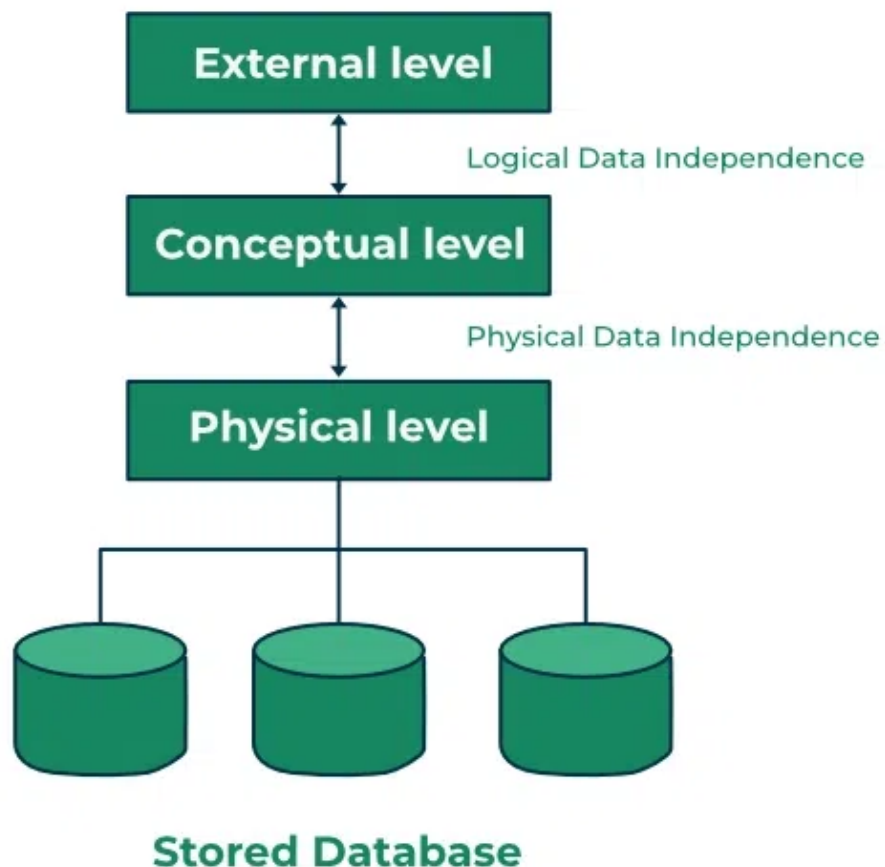
### Data Independence in DBMS

Data independence is a concept of DBMS which alters the schema of the database at one level of the database system without altering the schema definition of the same database at the next higher level. It helps the users by improving the quality of data. By using data independence, there is no need to modify the data structure in the application programs.

In the Database Management System, data independence is of two types:

1. **Logical Data Independence**
2. **Physical Data Independence**

## Data Independence in DBMS



## Logical Data Independence

Logical Data Independence is a term that changes the schema at the conceptual level without having to modify the external level or application program. The external or user view would not be affected when there is any change in conceptual view data.

This data independence is mainly used for separating the external schema from the schema at the conceptual level. Modifications at the logical level are compulsory when there is any modification in the logical structure of the database.

Following are the examples of logical data independence, which will not affect the external or user view:

1. We can easily break the existing record into more than two records.
2. We can also combine the two records into one.
3. We can easily modify, delete or insert the attribute.

## Physical Data Independence

Physical Data Independence is another type of data independence that alters the schema at the internal/physical level without having to alter the schema at the conceptual level. The conceptual structure of the database systems will not be affected if there is any modification in the storage size of the database.

This data independence is mainly used for separating the internal schema from the schema at the conceptual level. As compared to the above data independence, this data independence can be achieved easily.

Following are the examples of physical data independence, which will not affect the conceptual view:

1. We can modify the indexes in the database system.
2. We can easily modify the data structure, which is used for storage.

## 6. Explain in detail about different levels of abstraction in DBMS.

### Data Abstraction in DBMS

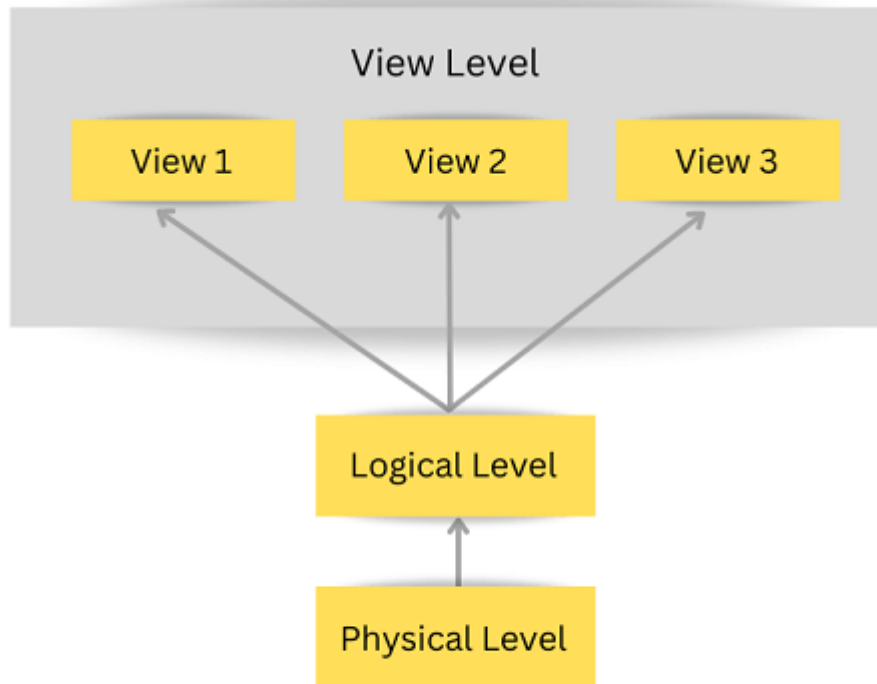
Data abstraction is the method of hiding the unimportant details that are present in the database from the end users to make the accessing of data easy and secure.

The main purpose of data abstraction is to hide irrelevant data and provide an abstract view of the data. With the help of data abstraction, developers hide irrelevant data from the user and provide them the relevant data. By doing this, users can access the data without any hassle, and the system will also work efficiently.

## Levels of Data Abstractions in DBMS

The data abstraction in DBMS is implemented in 3 layers:

- Physical or Internal Level
- Logical or Conceptual Level
- View or External Level



### Levels of Data Abstraction in DBMS

#### 1. Physical or Internal Level:

The physical or internal layer is the lowest level of data abstraction in the database management system. It is the layer that defines how data is actually stored in the database. It defines methods to access the data in the database. It defines complex data structures in detail, so it is very complex to understand, which is why it is kept hidden from the end user. Data Administrators (DBA) decide how to arrange data and where to store data. The Data Administrator (DBA) is the person whose role is to manage the data in the database at the physical or internal level. There is a data center that securely stores the raw data in detail on hard drives at this level.

#### 2. Logical or Conceptual Level:

The logical or conceptual level is the intermediate or next level of data abstraction. It explains what data is going to be stored in the database and what the relationship is between them. It describes the structure of the entire data in the form of tables. The logical level or conceptual level is less complex than the physical level. With the help of the logical level, Data Administrators (DBA) abstract data from raw data present at the physical level.

#### 3. View or External Level:

View or External Level is the highest level of data abstraction. There are different views at



this level that define the parts of the overall data of the database. This level is for the end-user interaction; at this level, end users can access the data based on their queries.

### Advantages of data abstraction in DBMS

- Users can easily access the data based on their queries.
- It provides security to the data stored in the database.
- Database systems work efficiently because of data abstraction.

## 7. Explain about Characteristics of DBMS.

### Characteristics of DBMS:

#### 1. Support for multiple views of data:

A database supports multiple views of data, allowing different users or groups to have tailored access to the database based on their needs.

#### 2. Sharing of data and multiuser system:

Modern database systems support multiple users accessing the same database simultaneously, facilitated by concurrency control strategies to maintain data integrity.

#### 3. Control of data redundancy:

Ideally, data redundancy is minimized to improve system performance, with each data item stored in only one place in the database.

#### 4. Data sharing:

Integration of data within a database system enables easy data sharing among employees and users, facilitating the generation of more information.

#### 5. Enforcement of integrity constraints:

DBMS enforces constraints to ensure data integrity, such as data type validation, uniqueness constraints, and access restrictions.

#### 6. Restriction of unauthorized access:

Different users have varying access privileges, managed by the security subsystem of a DBMS to control unauthorized access.

#### 7. Data independence:

DBMS allows for data independence by separating data descriptions from application programs, enabling changes to the data structure without affecting the programs.

#### 8. Transaction processing:

Concurrency control subsystems in DBMS ensure data consistency and validity during transaction processing, even with multiple users updating the same information.

#### 9. Provision for multiple views of data:

DBMS permits many users to access the database simultaneously without needing to know the underlying data storage details.

#### 10. Backup and recovery facilities:

DBMS provides backup and recovery methods to protect data from loss, with separate processes for data backup and recovery in case of system failures.



# UNIT - 2

## 1. Explain about Relational calculus with example queries.

### Relational Calculus

#### Procedural Language:

Languages that define how to get results from a database are called procedural languages. Relational algebra is a procedural language.

#### Declarative Language:

Languages that focus only on what to get from the database without specifying how to retrieve the results are called declarative languages. Relational Calculus is a declarative language.

Relational calculus is a non-procedural query language that uses mathematical predicate calculus.

In a database management system (DBMS), relational calculus focuses on "what you want" rather than "how to get it."

### Types of Relational Calculus:

#### 1. Tuple Relational Calculus (TRC):

- TRC is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables.
- It is declarative, specifying what data is required rather than how to retrieve it.
- TRC uses a tuple variable (t) to check if a predicate is true or false for each row of the table.
- Example Query: `{ t.Last_Name | Student(t) AND t.age > 30 }`
  - Result:

```
Last_Name
_____  
Singh
```

- Another Example Query: `{ t | Student(t) AND t.Last_Name = 'Singh' }`
  - Output:

```
First_Name Last_Name Age
_____  
_____
```

Ajeet	Singh	30
Chaitanya	Singh	31

## 2. Domain Relational Calculus (DRC):

- DRC filters records based on domains.
- Example Query:  $\{ \langle \text{First\_Name}, \text{Age} \rangle \mid \in \text{Student} \wedge \text{Age} > 27 \}$
- Output:

First_Name	Age
Ajeet	30
Chaitanya	31
Carl	28

## 2. Explain about Joins in DBMS with examples.

### JOINS in DBMS

Databases often consist of multiple tables, and JOINS are SQL constructions used to combine data from two or more tables. When you need to use columns from multiple tables in a result table, JOIN queries are the way to go. The JOIN keyword is utilized in SQL queries to join tables.

The syntax of an SQL JOIN is:

```
SELECT * FROM table1 JOIN table2 ON table1.id1=table2.id2
```

### Example:

#### orders table:

order_id	customer_id	order_date	order_amount
101	1	2024-04-01	100.00
102	2	2024-04-03	50.00
103	3	2024-04-04	75.00
104	1	2024-04-05	150.00

#### customers table:

id	name	email
1	John Smith	john@example.com

id	name	email
2	Jane Doe	jane@example.com
3	Robert Brown	robert@example.com

```
SELECT orders.order_id, orders.order_date, orders.order_amount,
       customers.name, customers.email
FROM orders
JOIN customers ON orders.customer_id = customers.id;
```

order_id	order_date	order_amount	name	email
101	2024-04-01	100.00	John Smith	john@example.com
102	2024-04-03	50.00	Jane Doe	jane@example.com
103	2024-04-04	75.00	Robert Brown	robert@example.com
104	2024-04-05	150.00	John Smith	john@example.com

## Types of JOIN

There are different types of JOINS that can be used in SQL:

- Inner Join
- Outer Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

### Inner Join:

An Inner Join retrieves data based on matched records according to the equality condition specified in the SQL query.

Syntax:

```
SELECT column-name-list FROM table-name1 INNER JOIN table-name2 WHERE table-
name1.column-name = table-name2.column-name;
```

Example:

```
SELECT * from class INNER JOIN class_info where class.id = class_info.id;
```

### Natural JOIN:

Natural Join is a type of Inner join which is based on columns having the same name and datatype present in both tables to be joined.

Syntax:

```
SELECT * FROM table-name1 NATURAL JOIN table-name2;
```

Example:

```
SELECT * from class NATURAL JOIN class_info;
```

## OUTER JOIN:

Outer Join includes both matched and unmatched data and further subdivides into:

### 1. Left Outer Join:

Returns a resultset table with matched data from two tables and then the remaining rows of the left table with null from the right table's columns.

Syntax:

```
SELECT column-name FROM table1 LEFT OUTER JOIN table-2 ON table-1.column-name = table-2.column-name;
```

Query:

```
SELECT * FROM song LEFT JOIN artist ON song.artist_id=artist.artist_id
```

### 2. Right Outer Join:

Returns a resultset table with matched data from the two joined tables, then the remaining rows of the right table, and null for the remaining columns of the left table.

Syntax:

```
SELECT column-name FROM table1 RIGHT OUTER JOIN table-2 ON table1.column-name = table2.column-name;
```

Query:

```
SELECT * FROM song RIGHT JOIN artist ON song.artist_id=artist.artist_id
```

### 3. Full Outer Join:

Returns a resultset table with the matched data of two tables and then the remaining rows of both the left and right tables.

Syntax:

```
SELECT column-name FROM table1 FULL OUTER JOIN table2 ON table1.column-name = table2.column-name;
```

Query:

```
SELECT * FROM song FULL JOIN artist ON song.artist_id=artist.artist_id
```

### 3. Explain Integrity Constraints in SQL with examples.

#### Integrity Constraints in DBMS

In database management systems (DBMS), a set of rules is used to maintain the quality and consistency of data in the database. Integrity constraints play a vital role in ensuring data integrity and preventing accidental damage to the database. There are four types of integrity constraints in DBMS:

##### 1. Domain Constraint:

- Every domain must contain atomic values (smallest indivisible units), meaning composite and multi-valued attributes are not allowed.
- Data type check ensures that values assigned to columns adhere to their specified data types.
- Example:

EID	Name	Phone
01	Bikash Dutta	123456789
NULL	NULL	234456678

Explanation: In the above relation, 'Name' is a composite attribute, and 'Phone' is a multi-valued attribute, violating domain constraint.

##### 2. Entity Integrity Constraint:

- Ensures that the primary key cannot be null, as the primary key is used to identify individual records in a table.
- Null values are allowed in other columns except the primary key column.
- Example:

EID	Name	Phone
01	Bikash	9000900099
02	Paul	600000009
NULL	Sony	9234567892

Explanation: In the above relation, 'EID' is the primary key and cannot take NULL values, violating entity integrity constraints.

##### 3. Referential Integrity Constraint:

- Ensures a valid relationship between two relational database tables.
- Confirms that a foreign key in one table references a corresponding value or attribute in another table, or is null.
- Example: If a foreign key in Table 1 refers to the Primary Key of Table 2, every value of the Foreign Key in Table 1 must be null or available in Table 2.

#### 4. Key Constraints:

- Keys are entities used to uniquely identify records within an entity set.
- Types of key constraints:
  - **NOT NULL:** Ensures a column does not hold NULL value.
  - **UNIQUE:** Enforces a column or set of columns to have unique values.
  - **CHECK:** Specifies a range of values for a particular column.
  - **DEFAULT:** Provides a default value to a column when no value is provided.
  - **PRIMARY KEY:** Uniquely identifies each record in a table, must have unique values, and cannot contain nulls. It combines the UNIQUE and NOT NULL constraints.
  - **FOREIGN KEY:** Columns of a table that point to the primary key of another table, acting as cross-references between tables. Prevents operations that can disrupt the link between tables.

**Note:** A foreign key of one table references the primary key of another table. Deleting data from the main table that points to data in another table results in an error - "Record in child table exists."

## 4. Explain in detail about Relational Algebra with examples.

### Relational Algebra in DBMS

- Relational algebra refers to a procedural query language that operates on relation instances, taking them as input and producing relation instances as output.
- Queries in relational algebra are performed using operators, which can be unary or binary.
- Programmers/users need to specify "What to Do" and "How to Do."
- Relational algebra follows a specific syntax that allows for easy access and retrieval of data from single or multiple table/data sources.

### Select ( $\sigma$ ):

- Selection operation is performed by the Selection Operator represented by "sigma" ( $\sigma$ ).
- It retrieves tuples (rows) from the table where the given condition is satisfied.
- It is a unary operator, requiring only one operand.

### Example:

Query Used:

```
σ Name and Age>21 (Student_Details)
```

ID	Name	Age	Marks
1	Alice	22	85
3	Carol	23	92
5	Eve	25	88

### Project (Π):

- Project operation is performed by the Projection Operator represented by "pi" (Π).
- It fetches tuples corresponding to a single attribute or multiple attributes.
- Also known as vertical partitioning as it separates the table vertically.
- It is a unary operator.

### Example:

Query Used:

```
ΠMarks(Student_Details)
```

Marks
85
78
92
75
88

### Union (∪):

- Union operation is performed by the Union Operator represented by "union" (∪).
- It selects all tuples from both relations, requiring the same set of attributes for both.
- It is a binary operator.

### Set Difference (-):

- Set Difference returns tuples from relation R that are not in relation S.
- It is denoted by a "Hyphen" (-) and is a binary operator.

### Example:

Query:

```
`Π Student_Name (STUDENT) - Π Student_Name (COURSE)
```

Student_Name
Alice
Frank
Grace

,

### Cartesian Product (X):

- Cartesian product combines tuples from two relations R and S.
- Denoted by the "X" symbol, it combines the information of two or more relations into a single relation.

#### Example:

Query:

R X S

### Intersection Operator ( $\cap$ ):

- Intersection operator selects common rows (tuples) from two tables/relations.
- Denoted by the  $\cap$  symbol, it returns tuples that are common to both relations.

#### Example:

Query:

```
 $\Pi$  Student_Name (COURSE)  $\cap$   $\Pi$  Student_Name (STUDENT)
```

Student_Name
Carol
Eve

## 5. Explain Key constraints in SQL with examples.

### Key Constraints:

Keys are entities used to uniquely identify an entity within its set. There could be multiple keys in a single entity set, but only one key is designated as the primary key. A primary key must contain unique and non-null values in the relational database table.

### NOT NULL:

NOT NULL constraint ensures that a column does not hold NULL values. By specifying this constraint, a particular column(s) cannot have NULL values.

#### Example:



```
CREATE TABLE STUDENT (  
    ID INT NOT NULL,  
    NAME VARCHAR(35) NOT NULL,  
    AGE INT NOT NULL,  
    PRIMARY KEY (ID)  
);
```

### UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values, meaning a particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT (  
    ID INT NOT NULL,  
    NAME VARCHAR(35) NOT NULL,  
    AGE INT,  
    EMAIL VARCHAR(50) UNIQUE,  
    PRIMARY KEY (ID)  
);
```

### CHECK:

CHECK constraint is used to specify a range of values for a particular column of a table, ensuring that the column must have values falling within the specified range.

```
CREATE TABLE STUDENT (  
    ID INT NOT NULL,  
    NAME VARCHAR(35) NOT NULL,  
    AGE INT CHECK (AGE ≥ 18 AND AGE ≤ 100),  
    PRIMARY KEY (ID)  
);
```

### DEFAULT:

DEFAULT constraint provides a default value to a column when no value is provided while inserting a record into a table.

```
CREATE TABLE STUDENT (  
    ID INT NOT NULL,  
    NAME VARCHAR(35) NOT NULL,  
    AGE INT DEFAULT 18,  
    PRIMARY KEY (ID)  
);
```

### PRIMARY KEY:

Primary key uniquely identifies each record in a table, containing unique values and cannot hold nulls. It has automatically applied UNIQUE and NOT NULL constraints and is usually

used to index the table or uniquely identify each tuple.

```
CREATE TABLE STUDENT (  
    ID INT NOT NULL,  
    NAME VARCHAR(35) NOT NULL,  
    AGE INT,  
    PRIMARY KEY (ID)  
);
```

### FOREIGN KEY Constraint:

Foreign keys are columns in a table that point to the primary key of another table, establishing a cross-reference between tables. FOREIGN KEY constraint prevents operations that can disrupt the relationship between the tables. It ensures that data integrity is maintained and is used when deletion from one table should not impact data in another table.

```
CREATE TABLE CLASS (  
    CLASS_ID INT NOT NULL,  
    CLASS_NAME VARCHAR(50),  
    STUDENT_ID INT,  
    PRIMARY KEY (CLASS_ID),  
    FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT(ID)  
);
```

One crucial aspect to note is that the Foreign Key of one table references the Primary Key of another table. Deleting data from the main table that refers to data in another table will result in an error - "Record in child table exists."

## 6. What is a view? How to specify a view? Write about view implementation techniques with example queries.

A **view** in a database management system (DBMS) is a virtual table created by a query that selects data from one or more underlying tables. It does not store data itself but provides a way to present data in a particular format or with specific filters. Views can simplify queries and enhance security by limiting access to specific columns or rows of data.

### Specifying a View

You can specify a view using the `CREATE VIEW` statement in SQL. This statement allows you to define the view by providing a name and a SELECT query that specifies which columns and rows from the underlying tables should be part of the view.

Here's the syntax for creating a view:

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...
```

```
FROM table_name  
WHERE condition;
```

For example:

```
CREATE VIEW employee_view AS  
SELECT employee_id, name, department  
FROM employees  
WHERE department = 'Sales';
```

This creates a view called `employee_view` that selects `employee_id`, `name`, and `department` from the `employees` table, only including employees in the 'Sales' department.

## View Implementation Techniques

There are several techniques for implementing views:

1. **Query Rewriting:** When a query involves a view, the DBMS rewrites the query using the view's definition. For example, if you query the `employee_view` for employee names, the DBMS will execute the underlying query defined in the view and then apply additional filters, if any.
2. **Materialized Views:** In some DBMS, views can be materialized, which means the query results are stored in a table. This can improve performance for complex queries that access the same view repeatedly.
3. **Access Control and Security:** Views can be used to control access to data by restricting which columns or rows are visible to a user. For example, a view could provide access only to specific columns of a table, hiding sensitive information.

## Example Queries Using Views

Here are some examples of queries using views:

1. **Select Data from a View:**

```
SELECT * FROM employee_view;
```

2. **Join a View with Another Table:**

```
SELECT e.name, d.department_name  
FROM employee_view e  
JOIN departments d ON e.department = d.department_id;
```

3. **Use a View in a WHERE Clause:**

```
SELECT * FROM orders  
WHERE customer_id IN (
```

```
SELECT customer_id FROM customer_view
WHERE location = 'New York'
);
```

## Inserting, Deleting, and Updating Data in Views

- **Inserting and Deleting Rows:** Rows can be inserted or deleted from updatable views just like from tables. However, whether a view is updatable depends on the complexity of the underlying query.
- **Updating Views:** Views can be updated using the `CREATE OR REPLACE VIEW` statement. This allows you to modify the view's query or add/remove columns.

```
CREATE OR REPLACE VIEW employee_view AS
SELECT employee_id, name, salary
FROM employees
WHERE salary > 50000;
```

## WITH CHECK OPTION

The `WITH CHECK OPTION` clause ensures that any changes made to a view must satisfy the view's conditions. This clause is useful for updatable views.

For example:

```
CREATE VIEW employee_view AS
SELECT employee_id, name, department
FROM employees
WHERE department = 'Sales'
WITH CHECK OPTION;
```

This view can only accept insertions or updates that satisfy the condition `department = 'Sales'`.

## UNIT - 3

### 1. Explain DDL, DML, and TCL commands with example Queries.

#### Data Definition Language (DDL)

DDL commands are used to create and modify the structure of a database and its objects such as tables. The main DDL commands are:

1. **CREATE:** Creates new database objects such as databases and tables.
  - Example for creating a database:

```
CREATE DATABASE database_name;
```

- Example for creating a table:

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

2. **ALTER**: Modifies the structure of an existing database object, such as adding or changing columns in a table.

- Example for adding a column:

```
ALTER TABLE Student ADD Ph_number INTEGER(12);
```

- Example for modifying a column's datatype:

```
ALTER TABLE Student MODIFY City VARCHAR(50);
```

3. **RENAME**: Changes the name of an existing database object.

- Example:

```
ALTER TABLE Student RENAME TO Student_details;
```

4. **DROP**: Permanently removes a database object.

- Example:

```
DROP TABLE Student;
```

5. **TRUNCATE**: Removes all rows from a table, but the table itself remains.

- Example:

```
TRUNCATE TABLE Student;
```

## Data Manipulation Language (DML)

DML commands are used to manipulate data within tables, such as inserting, updating, or deleting rows. The main DML commands are:

1. **INSERT**: Inserts new rows into a table.

- Example:

```
INSERT INTO Student VALUES (01, 'Rohini', 20, 'Delhi');
```

2. **UPDATE**: Modifies existing rows in a table.

- Example:

```
UPDATE Student SET Name='Mahima' WHERE Roll_number=03;
```

3. **DELETE**: Removes specific rows from a table based on a condition.

- Example:

```
DELETE FROM Student WHERE Roll_number=2;
```

## Transaction Control Language (TCL)

TCL commands manage the changes made by DML statements, ensuring data integrity and consistency. The main TCL commands are:

1. **COMMIT**: Saves all changes made during a transaction.

- Example:

```
INSERT INTO Student_details VALUES(02, 'Rajat', 21, 'Delhi',  
9874008);  
COMMIT;
```

2. **ROLLBACK**: Undoes any changes made during a transaction, reverting the state to the last commit or rollback point.

- Example:

```
DELETE FROM Student_details WHERE Roll_number=4;  
ROLLBACK;
```

3. **SAVEPOINT**: Creates a named point in a transaction that allows you to rollback to that point.

- Example:

```
SAVEPOINT SP1;  
INSERT INTO Student_details VALUES(05, 'Suraj', 21, 'Goa',  
9974458);  
SAVEPOINT SP2;  
ROLLBACK TO SP1;
```

## 2. Explain about Aggregate Functions in DBMS with example queries.

Aggregate functions in SQL are used to perform calculations based on multiple rows and return a single value according to the given query. They are commonly used with SELECT statements to summarize data and can be combined with GROUP BY and HAVING clauses to group and filter results.

Main types of aggregate functions in SQL:

### COUNT()

The `COUNT()` function is used to count the number of rows in a database table. It can count both numeric and non-numeric data types, and it considers duplicate and NULL values.

- Syntax: `COUNT(*)` or `COUNT([ALL | DISTINCT] expression)`
- Example with `GROUP BY` clause:

```
SELECT city, COUNT(*) FROM employees GROUP BY city;
```

- Example with `HAVING` and `ORDER BY` clauses:

```
SELECT name, COUNT(age) FROM employees GROUP BY age HAVING COUNT(age) ≥ 2 ORDER BY COUNT(age);
```

### SUM()

The `SUM()` function returns the sum of all non-NULL values in a specified column. It only works on numeric fields.

- Syntax: `SUM()` or `SUM([ALL | DISTINCT] expression)`
- Example with `GROUP BY` clause:

```
SELECT employee_id, SUM(working_hours) FROM work_hours GROUP BY employee_id;
```

- Example with `HAVING` clause:

```
SELECT occupation, SUM(working_hours) FROM employees GROUP BY occupation HAVING SUM(working_hours) > 24;
```

### AVG()

The `AVG()` function returns the average of all non-NULL values in a specified column. It works only on numeric fields.

- Syntax: `AVG()` or `AVG([ALL | DISTINCT] expression)`
- Example query:

```
SELECT AVG(daily_typing_pages) FROM employee_tbl;
```

- Example with `GROUP BY` clause:

```
SELECT name, AVG(daily_typing_pages) FROM employee_tbl GROUP BY name;
```

## MIN()

The `MIN()` function returns the minimum value present in a specified column. If no rows are selected, it returns `NULL`.

- Syntax: `MIN()` or `MIN([ALL | DISTINCT] expression)`
- Example query:

```
SELECT MIN(daily_typing_pages) FROM employee_tbl;
```

- Example with `GROUP BY` clause:

```
SELECT id, name, MIN(daily_typing_pages) FROM employee_tbl GROUP BY name;
```

## MAX()

The `MAX()` function returns the maximum value present in a specified column. If no rows are selected, it returns `NULL`.

- Syntax: `MAX()` or `MAX([ALL | DISTINCT] expression)`
- Example query:

```
SELECT MAX(daily_typing_pages) FROM employee_tbl;
```

- Example with `GROUP BY` clause:

```
SELECT id, name, MAX(daily_typing_pages) FROM employee_tbl GROUP BY name;
```



### 3. Explain about Nested Queries in DBMS with example Queries.

Nested queries, also known as subqueries, are queries within another SQL query. The inner query, or subquery, is used to return data that the outer query, or main query, uses as a condition to further restrict the data to be retrieved. Subqueries can be used in SELECT, INSERT, UPDATE, and DELETE statements.

#### Types of Nested Queries in SQL:

##### 1. Independent Nested Queries

- In independent nested queries, the execution order is from the innermost query to the outer query.
- The outer query uses the result of the inner query.
- Operators such as `IN`, `NOT IN`, `ALL`, and `ANY` are used in independent nested queries.

##### 2. Correlated Nested Queries

- In correlated nested queries, the inner query uses values from the outer query to execute the inner query for every row processed by the outer query.
- These queries run more slowly as the inner query executes for every row of the outer query's result.

#### Example Queries:

##### 1. Independent Nested Query with `IN` operator:

- This example retrieves all customers from the `customers` table where their ID is present in the subquery result, which returns IDs of customers with a salary greater than 4500.
- Example:

```
SELECT * FROM customers WHERE id IN (SELECT id FROM customers WHERE salary > 4500);
```

##### 2. Correlated Nested Query:

- This example deletes records from the `customers` table for customers whose age is greater than or equal to 27 and who have corresponding records in the `customers_bkp` table.
- Example:

```
DELETE FROM customers WHERE age IN (SELECT age FROM customers_bkp WHERE age ≥ 27);
```

##### 3. Subquery with the INSERT statement:

- This example inserts data into the `customers_bkp` table from the `customers` table where the IDs match.
- Example:

```
INSERT INTO customers_bkp SELECT * FROM customers WHERE id IN  
(SELECT id FROM customers);
```

#### 4. Subquery with the UPDATE statement:

- This example updates the salary of customers in the `customers` table to be 0.25 times the current salary for customers whose age is greater than or equal to 27.
- Example:

```
UPDATE customers SET salary = salary * 0.25 WHERE age IN (SELECT  
age FROM customers_bkp WHERE age ≥ 27);
```