What is a database?

A **database** ek digital system hai jo data ko store, organize, aur manage karta hai taki usko easily access aur update kiya ja sake. Yeh ek tarah ka computerized record-keeping system hota hai jo different types ki information ko efficiently handle karta hai.

Why is a database useful?

- **Data storage** – Information safe aur organized rehti hai.

- **Fast access** – Zaroori data ko jaldi search aur retrieve kiya ja sakta hai.

- **Easy management** – Data ko modify, delete aur update karna simple hota hai.

- **Security** – Important information secure rehti hai, sirf authorized users access kar sakte hain.

Types of Databases:

1. **Relational Database (SQL)** – Structured format mein tables ke form mein data store hota hai. *(Example: MySQL, PostgreSQL, SQLite)*

2. **NoSQL Database** – Flexible aur schema-free hote hain, mostly big data aur real-time applications ke liye use hote hain. *(Example: MongoDB, Firebase)*

3. **Cloud Database** – Online hosted hote hain jo internet se access kiye ja sakte hain. *(Example: Google Cloud Firestore, AWS RDS)*

4. **Graph Database** – Nodes aur relationships ke form mein data store hota hai, jo complex connections handle karne ke liye useful hota hai. *(Example: Neo4j)*

5. **In-Memory Database** – Super-fast storage jo RAM mein data rakhta hai, mostly real-time processing ke liye use hota hai. *(Example: Redis)*

What is DBMS?

A **Database Management System (DBMS)** ek software hai jo databases ko create, manage, aur control karne ke liye use hota hai. Yeh users ko data store, retrieve, update, aur delete karne ki facility deta hai.

🔹 **Examples**: MySQL, PostgreSQL, MongoDB, Oracle

DBMS vs File System

Feature	DBMS	File System
Data Organization	Structured (Tables, Relations)	Unstructured (Files, Folders)
Data Security	High (Access Control)	Low (Limited Security)
Data Integrity	Maintained (Constraints)	No built-in integrity check
Data Redundancy	Reduced (Normalization)	High (Duplicate Files)
Data Access	Fast (Queries)	Slow (Manual Search)
Multi-User Support	Yes	Limited

Main Difference

File system sirf data store karta hai, jabki **DBMS data ko organize, secure, aur efficiently manage karta hai**.

DBMS (Database Management System) ke use karne ke kai **advantages** hain:

1. **Data Redundancy Control**

DBMS me data ko organize karke redundancy ko minimize kiya jata hai, yani ek hi data ka duplicate version nahi hota.

2. **Data Integrity**

DBMS me **data consistency** aur accuracy maintain hoti hai, kyunki constraints aur rules lagaye jaate hain.

3. **Data Security**

DBMS me authorized users ko access diya jaata hai aur sensitive data ko protect karne ke liye encryption aur authentication features hote hain.

4. **Easy Data Access**

DBMS me **structured queries (SQL)** ke through data ko easily search, retrieve, aur update kiya ja sakta hai.

5. **Multi-User Support**

DBMS multiple users ko ek saath data access karne ka option deta hai without affecting the performance.

6. **Backup & Recovery**

Agar data loss hota hai toh DBMS me automatic backup aur recovery features hote hain, jisse data safe rehta hai.

7. **Data Independence**

DBMS me data aur applications ke beech independence hoti hai, yani data ko change kiya ja sakta hai bina application code ko modify kiye.

8. **Efficient Data Management**

DBMS large amounts of data ko efficiently manage karne

ki capacity rakhta hai, jisse data retrieval fast aur easy hota hai.

In short, **DBMS** helps in **data organization**, **security**, **accessibility**, aur **efficiency**, making it essential for businesses and applications.

7. Data Independence

DBMS me **data aur application** ke beech independence hoti hai, yani data ko change kiya ja sakta hai bina application code ko modify kiye.

How?

- **Physical Data Independence**: Data ko physically store karne ki method ko change kiya jaa sakta hai bina application ko affect kiye.

- **Logical Data Independence**: Data structure ko change kiya jaa sakta hai (jaise columns ya tables modify karna), bina application code ko touch kiye.

Isse, **DBMS** data aur application ke beech

abstraction provide karta hai, jo flexibility aur efficiency badhata hai.

Types of Databases:

1. **Relational Database (SQL)**

   - Data tables mein store hoti hai, aur **Structured Query Language (SQL)** ka use hota hai.

   - Example: **MySQL, PostgreSQL, Oracle**

2. **NoSQL Database**

   - Schema-free aur flexible hota hai, mostly large-scale data aur real-time applications ke liye.

   - Example: **MongoDB, Firebase**

3. **Cloud Database**

   - Online hosted hoti hai, jo internet se access ki ja sakti hai.

   - Example: **Google Cloud Firestore, AWS RDS**

4. **Graph Database**

   - Data ko nodes aur relationships ke form mein store kiya jata hai, jo complex connections ko handle karta hai.

   - Example: **Neo4j**

5. **In-Memory Database**

   - Data ko RAM mein store karta hai, jo super-fast access provide karta hai.

   - Example: **Redis**

6. **Object-Oriented Database**

   - Data ko objects ki form mein store kiya jata hai, jo object-oriented programming languages se closely related hota hai.

   - Example: **ObjectDB**

Each type has its own use case depending on the data requirements and application needs.

A **relational database** ek type ka database hota hai jo

tables (ya relations) mein data ko store karta hai. Har table rows aur columns se bana hota hai, jahan rows ko **records** aur columns ko **fields** kaha jata hai.

Key Features of a Relational Database:

1. **Tables** – Data ko structured format mein tables mein store kiya jata hai.

2. **Primary Key** – Har record ko uniquely identify karne ke liye ek primary key hoti hai.

3. **Foreign Key** – Tables ke beech relationships establish karne ke liye foreign keys use hoti hain.

4. **SQL** – Data ko manage karne ke liye **Structured Query Language (SQL)** ka use hota hai.

5. **Normalization** – Data redundancy ko reduce karne ke liye normalization techniques use ki jati hain.

Examples:

- **MySQL**

- **PostgreSQL**

- **Oracle**

SQL (Structured Query Language) ek standard programming language hai jo **relational databases** ko manage aur manipulate karne ke liye use hoti hai. Iska use data ko retrieve (select), insert, update, aur delete karne ke liye hota hai.

Key Features of SQL:

1. **Data Retrieval** – SQL ka use data ko search aur retrieve karne ke liye hota hai.

   Example: `SELECT * FROM Students;`

2. **Data Insertion** – SQL ka use naye data ko insert karne ke liye hota hai.

   Example: `INSERT INTO Students (StudentID, Name, Age) VALUES (4, 'Sara Khan', 23);`

3. **Data Update** – SQL ka use existing data ko modify karne ke liye hota hai.

   Example: `UPDATE Students SET Age = 24 WHERE

StudentID = 4;`

4. **Data Deletion** – SQL ka use data ko delete karne ke liye hota hai.

   Example: `DELETE FROM Students WHERE StudentID = 4;`

5. **Creating and Modifying Tables** – SQL ka use tables ko create aur modify karne ke liye bhi hota hai.

   Example: `CREATE TABLE Courses (CourseID INT, CourseName VARCHAR(100));`

Common SQL Commands:

- `SELECT`: Data retrieve karna
- `INSERT`: Data insert karna
- `UPDATE`: Data update karna
- `DELETE`: Data delete karna
- `CREATE`: Table create karna
- `ALTER`: Table modify karna

- `DROP`: Table delete karna

SQL relational databases ke saath interaction ke liye ek essential tool hai.

DROP and ALTER in SQL:

- **`ALTER`**: Table ko modify karne ke liye use hota hai. Aap columns add, modify, ya delete kar sakte ho.
  - Example: `ALTER TABLE Students ADD Address VARCHAR(255);` (Add new column)

- **`DROP`**: Table ya column ko permanently delete karne ke liye use hota hai.
  - Example: `DROP TABLE Students;` (Delete entire table)

`VARCHAR(100)`:

- **`VARCHAR`** ek data type hai jo text store karta hai.

- **`100`** ka matlab hai ki maximum **100 characters** tak ki text store ki ja sakti hai.

  - Example: `VARCHAR(100)` means you can store a text of up to 100 characters.

Entity-Relationship (ER) model, ya **Entity-Relationship Diagram (ERD)**, ek visual representation hota hai database ki structure ka, jo dikhata hai ki alag-alag **entities** (objects ya concepts) kaise connected aur interact karte hain **relationships** ke through, symbols jaise rectangles, diamonds, aur connecting lines ka use karke.

Key Concepts of ER Model:

1. **Entity**:

   - **Entity** kisi bhi object ya concept ko represent karta hai, jiska data database mein store hota hai.

   - Example: **Student**, **Employee**, **Course**

2. **Attribute**:

   - **Attribute** ek property ya characteristic hoti hai kisi entity ki.

   - Example: **Name**, **Age**, **Address** (for **Student**)

3. **Relationship**:

   - **Relationship** dikhata hai ki entities kaise ek dusre se related hoti hain.

   - Example: Ek **Student** ek **Course** mein enroll hota hai.

Types of Relationships:

1. **One-to-One (1:1)**:

   - Ek entity dusre entity ke saath ek hi relation rakhti hai.

   - Example: Ek **Employee** ka ek hi **EmployeeID** hota hai.

2. **One-to-Many (1:N)**:

- Ek entity ka multiple relations dusre entity ke saath hota hai.

- Example: Ek **Department** ke andar kai **Employees** ho sakte hain.

3. **Many-to-Many (M:N)**:

- Multiple entities ka relation multiple dusre entities ke saath hota hai.

- Example: **Students** kai **Courses** mein enroll karte hain, aur **Courses** mein kai **Students** hote hain.

ER Diagram Example:

- **Entities**: **Student**, **Course**

- **Attributes**: **Student** ka **Name**, **Age**; **Course** ka **CourseName**, **Credits**

- **Relationship**: **Student** **Course** mein enroll hota hai.

ER model help karta hai data ko visualize karne mein aur entities ke beech ke connections ko samajhne mein, jo database ko design aur organize karne mein asaan bana deta hai.

Normalization ek process hai jisme hum database ko organize karte hain taaki data redundancy (duplicate data) ko minimize kiya ja sake aur data consistency maintain ho. Iska purpose hota hai tables ko aise design karna ki unmein **anomalies** (insertion, deletion, aur update anomalies) na ho.

Why is Normalization Important?

1. **Minimizes Data Redundancy**:

   Normalization duplicate data ko eliminate karta hai, jisse storage efficient hota hai aur data consistency maintain rehti hai.

2. **Improves Data Integrity**:

   Jab data redundant nahi hota, toh kisi bhi data ko

update karne ya delete karne mein mistake hone ka chance kam ho jata hai.

3. **Avoids Anomalies**:

Normalization insertion, deletion, aur update anomalies ko avoid karta hai. For example, agar ek record ko delete karte ho, toh related data bhi delete ho jata hai, jo unwanted ho sakta hai.

4. **Efficient Queries**:

Normalized database mein queries run karna easy aur fast hota hai, kyunki data properly organized hota hai.

Normalization Process:

Normalization typically multiple stages mein hota hai, jinhe **normal forms** (1NF, 2NF, 3NF, etc.) kehte hain:

1. **1st Normal Form (1NF)**:

- **Atomic** banane ka matlab hai ki har column mein sirf ek value honi chahiye (no repeating groups). Yani, har

cell mein indivisible (atomic) data hona chahiye.

- Example: Agar ek column mein **multiple phone numbers** diye hain, toh usse split karke **alag-alag rows** mein rakhna hoga.

2. **2nd Normal Form (2NF)**:

- **Partial Dependency** ka matlab hai ki non-key attributes **only part** of the primary key par depend na ho. Agar primary key **composite (multiple attributes)** hai, toh non-key attributes ko **complete primary key** par depend karna chahiye.

- Example: Agar **StudentID** aur **CourseID** milke primary key banate hain, toh **StudentName** sirf **StudentID** par depend kar raha ho, yeh partial dependency hai. Isse remove karke **Student** ko alag table mein store karna hoga.

3. **3rd Normal Form (3NF)**:

- **Transitive Dependency** ka matlab hai ki non-key attribute kisi **non-key attribute** par depend na kare. Sirf primary key ke upar depend hona chahiye.

- Example: Agar **StudentID** ke through aapka **StudentName** milta hai aur **StudentName** ke through **City** milta hai, toh **City** ko **StudentName** ke through store karna transitive dependency hai. Yeh remove karke **City** ko **Student** table ke attributes ke saath store karna hoga.

, toh **City** ko **StudentID** ke directly depend hona chahiye, na ki **StudentName** ke.

In short, **Normalization** ka main goal hota hai data ko properly structure karna, taaki database ki efficiency, consistency aur integrity maintain ho sake.

Primary Keys aur **Foreign Keys** relational databases mein key concepts hain jo data ko uniquely identify karte hain aur tables ke beech relationships establish karte hain.

Primary Key:

- **Primary key** ek unique identifier hota hai har record (row) ke liye table mein.

- Har table mein ek hi **primary key** hoti hai.

- **Primary key** ko use karke hum kisi bhi record ko uniquely identify kar sakte hain, aur is key ke under **duplicate values** allowed nahi hote.

- **Null values** bhi allowed nahi hoti primary key mein.

Example:

- Agar humare paas ek **Student** table hai, toh **StudentID** ko primary key banaya ja sakta hai, kyunki har student ka ID unique hoga.

StudentID (Primary Key)	Name	Age
1	John	20
2	Sara	22
3	Mike	21

Yahan **StudentID** ek primary key hai, jo har student ko uniquely identify karti hai.

Foreign Key:

- **Foreign key** ek attribute (ya set of attributes) hota hai jo **ek table ko doosre table ke saath link karta hai**.

- **Foreign key** us table ki **primary key** ko refer karta hai jisme relationship establish ho rahi hoti hai.

- Foreign key ki madad se hum tables ke beech relationships bana sakte hain, jaise **one-to-many** ya **many-to-many** relationships.

Example:

- Agar humare paas ek **Student** table aur ek **Course** table hai, aur har student ka ek course mein enrollment hai, toh **CourseID** ko foreign key bana sakte hain, jo **Course** table ki primary key ko refer karega.

Student Table:

StudentID (Primary Key)	Name	CourseID (Foreign Key)
1	John	101
2	Sara	102
3	Mike	101

Course Table:

CourseID (Primary Key)	CourseName
101	Math
102	Science

Yahan **CourseID** ek foreign key hai **Student** table mein, jo **Course** table ki primary key ko refer karta hai.

Key Differences:

Feature	Primary Key	Foreign Key
Purpose	Uniquely identifies a record	Creates a relationship between tables
Uniqueness	Values must be unique	Values can repeat
Null Values	Cannot be null	Can have null values
Table	Belongs to one table	Belongs to the referencing table

In short, **Primary Key** record ko uniquely identify karta hai, aur **Foreign Key** doosre table ki primary key ko refer karke tables ke beech relationship establish karta hai.

DDL, **DML**, aur **DCL** SQL (Structured Query Language) ke different types of commands hain jo database operations ko manage karte hain. Inka kaam data ko define, manipulate, aur control karna hai. Let’s explore them in short:

1. **DDL (Data Definition Language):**

DDL commands ka use database aur tables ka structure define karne ke liye hota hai, jaise tables create karna, modify karna, ya delete karna.

- **Common DDL Commands:**
  - **CREATE**: Naye database ya table ko create karta hai.
  - **ALTER**: Already existing table ko modify karta hai (column add, modify, ya delete).
  - **DROP**: Database ya table ko delete karta hai.
  - **TRUNCATE**: Table ke sare records ko remove karna (structure ko nahi).

Examples:

- `CREATE TABLE Students (StudentID INT, Name VARCHAR(50));`

  (New table **Students** create kar raha hai.)

- `ALTER TABLE Students ADD Age INT;`

  (Table me **Age** column add kar raha hai.)

- `DROP TABLE Students;`

  (Table **Students** ko delete kar raha hai.)

2. **DML (Data Manipulation Language):**

DML commands ka use actual data ko manipulate karne ke liye hota hai, jaise data ko insert karna, update karna, ya delete karna.

- **Common DML Commands:**

  - **SELECT**: Data ko retrieve (read) karta hai.

  - **INSERT**: Naye data ko insert karta hai.

- **UPDATE**: Existing data ko modify karta hai.

- **DELETE**: Data ko delete karta hai.

Examples:

- `SELECT * FROM Students;`

  (All records ko **Students** table se retrieve kar raha hai.)

- `INSERT INTO Students (StudentID, Name, Age) VALUES (1, 'John', 20);`

  (Naye record ko **Students** table mein insert kar raha hai.)

- `UPDATE Students SET Age = 21 WHERE StudentID = 1;`

  (Record ko update kar raha hai jisme **StudentID** 1 hai.)

- `DELETE FROM Students WHERE StudentID = 1;`

  (Record ko delete kar raha hai jisme **StudentID** 1 hai.)

3. **DCL (Data Control Language):**

DCL commands ka use database access control ko manage karne ke liye hota hai, jaise permissions dena ya revoke karna.

- **Common DCL Commands:**
  - **GRANT**: User ko specific permissions dega (jaise SELECT, INSERT, UPDATE).
  - **REVOKE**: User ke permissions ko remove karna.

Examples:

- `GRANT SELECT ON Students TO User1;`

  (User1 ko **Students** table par **SELECT** permission de raha hai.)

- `REVOKE DELETE ON Students FROM User1;`

  (User1 se **DELETE** permission ko revoke kar raha hai.)

Summary Table:

Type	Full Form	Purpose	Example Command
DDL	Data Definition Language	Defines the database structure (tables, columns, etc.)	`CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`
DML	Data Manipulation Language	Manages data inside tables (insert, update, delete)	`SELECT`, `INSERT`, `UPDATE`, `DELETE`
DCL	Data Control Language	Controls access permissions	`GRANT`, `REVOKE`

In short:

- **DDL**: Database structure define karta hai.
- **DML**: Database mein data ko manipulate karta hai.

- **DCL**: Permissions ko manage karta hai.

DELETE aur **TRUNCATE** dono SQL commands hain jo data ko remove karte hain, lekin in dono mein kuch important differences hote hain:

1. DELETE:

- **Functionality**: DELETE command table ke rows ko remove karta hai, lekin **structure** ko unaffected rakhta hai (table ka structure, columns, aur constraints remain the same).

- **Row-by-Row Deletion**: DELETE command har row ko ek-ek karke delete karta hai, isliye ismein time zyada lag sakta hai.

- **Rollback**: DELETE command ko rollback kiya ja sakta hai (agar transaction ka use kiya gaya ho).

- **Where Clause**: DELETE mein aap **WHERE** clause use kar sakte hain, jisse aap specific rows ko delete kar sakte hain.

- **Triggers**: DELETE operation triggers ko activate kar

sakta hai (agar triggers defined hain).

Example:

```sql
DELETE FROM Students WHERE StudentID = 1;
```

Isse **StudentID = 1** wale row ko delete kiya jayega.

2. TRUNCATE:

- **Functionality**: TRUNCATE command table ke sare rows ko remove karta hai, lekin table ka structure, columns, aur constraints unaffected rehte hain.

- **All Rows Removed**: TRUNCATE command **sabhi rows ko ek sath delete karta hai** (no row-by-row operation), isliye yeh faster hota hai.

- **No WHERE Clause**: TRUNCATE mein **WHERE** clause nahi hota, matlab yeh poore table ko clean kar deta hai.

- **Rollback**: TRUNCATE ko rollback nahi kiya ja sakta

hai (agar transaction ka use na ho).

- **Triggers**: TRUNCATE operation triggers ko activate nahi karta hai.

Example:

```sql
TRUNCATE TABLE Students;
```

Isse **Students** table ke sabhi rows delete ho jayenge.

Key Differences:

Feature	DELETE	TRUNCATE
Operation	Row-by-row deletion	

All rows deleted at once |

| **Use of WHERE Clause** | Yes | No |

| **Rollback** | Can be rolled back (if in transaction) | Cannot be rolled back |

| **Triggers** | Activates triggers (if defined) | Does not activate triggers |

| **Speed** | Slower (due to row-by-row deletion) | Faster (because it deletes all rows in one go) |

| **Affects Structure** | No (structure remains the same) | No (structure remains the same) |

In short:

- **DELETE**: Row-by-row operation, slow, allows WHERE clause, can be rolled back.

- **TRUNCATE**: Faster, deletes all rows, no WHERE clause, cannot be rolled back.

ROLL BACK IS FOR UNDO

Constraints SQL mein rules hote hain jo table ke data ko valid aur consistent banaye rakhte hain. Ye ensure karte hain ki data properly stored ho aur integrity maintain ho.

Types of Constraints:

1. **PRIMARY KEY**:
   - Unique identifier hota hai, aur NULL nahi ho sakta.
   - Example: `StudentID INT PRIMARY KEY`

2. **FOREIGN KEY**:
   - Ek table ko doosre table se link karta hai.
   - Example: `FOREIGN KEY (StudentID) REFERENCES Students(StudentID)`

3. **UNIQUE**:
   - Column mein duplicate values allowed nahi hoti.

- Example: `Email VARCHAR(100) UNIQUE`

4. **NOT NULL**:

- Column mein NULL value nahi ho sakti.

- Example: `Name VARCHAR(50) NOT NULL`

5. **CHECK**:

- Data insert/update karte waqt condition apply karta hai.

- Example: `Age INT CHECK (Age >= 18)`

6. **DEFAULT**:

- Agar value na diya ho to default value set karta hai.

- Example: `Status VARCHAR(20) DEFAULT 'Active'`

In short, **constraints** SQL mein data integrity ko maintain karne ke liye use hote hain.

Before and **After** grouping ka matlab hai ki

WHERE aur **HAVING** ka use kis stage par hota hai jab data ko group kiya ja raha ho.

Before Grouping (WHERE):

- Jab aap data ko **filter** karte hain **GROUP BY** ke pehle, to aap **WHERE** ka use karte hain.

- **WHERE** condition ko **raw data** par apply kiya jata hai, jo grouping ke liye use ho raha hota hai.

Example:

`WHERE Age > 18`

Ye filter karta hai, sirf wo rows jinme Age 18 se zyada ho, aur phir un rows ko group kiya jata hai.

After Grouping (HAVING):

- Jab data ko **group** kar liya jata hai, aur uske baad aapko group ko **filter** karna ho, to aap **HAVING** ka use karte hain.

- **HAVING** **grouped results** ko filter karta hai jo **aggregate functions** (jaise COUNT, SUM) ke baad

evaluate hota hai.

Example:

`HAVING COUNT(*) > 10`

Ye filter karega, sirf wo groups jinmein 10 se zyada items ho.

Summary:

- **WHERE**: Data ko **before grouping** filter karta hai.

- **HAVING**: Data ko **after grouping** filter karta hai.

JOINs SQL mein ek important concept hai, jo ek ya zyada tables ke data ko combine karta hai based on a related column. Jab aapko multiple tables se data fetch karna hota hai, tab JOINs ka use hota hai.

Types of JOINs in SQL:

1. **INNER JOIN**:

   - Ye **common rows** ko select karta hai jo dono tables mein match karte hain.

   - Agar ek table mein row nahi hai jo doosre table se match karti ho, to wo row result mein nahi aayegi.

   Example:

   ```sql

   SELECT Employees.Name, Departments.DepartmentName

   FROM Employees

   INNER JOIN Departments

   ON Employees.DepartmentID = Departments.DepartmentID;

   ```

2. **LEFT JOIN (or LEFT OUTER JOIN)**:

   - Ye **left table** ki saari rows ko select karta hai, chahe wo right table mein match kare ya nahi.

- Agar right table mein match nahi milta, to **NULL** values return hoti hain.

Example:

```sql
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

3. **RIGHT JOIN (or RIGHT OUTER JOIN)**:

- Ye **right table** ki saari rows ko select karta hai, chahe wo left table mein match kare ya nahi.

- Agar left table mein match nahi milta, to **NULL** values return hoti hain.

Example:

```sql
SELECT Employees.Name,
Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments
ON Employees.DepartmentID =
Departments.DepartmentID;
```

4. **FULL JOIN (or FULL OUTER JOIN)**:

- Ye dono tables ki saari rows ko select karta hai, chahe unmein match ho ya na ho.

- Agar kisi table mein row match nahi karti, to wo **NULL** value ke saath show hoti hai.

Example:

```sql
SELECT Employees.Name,
```

Departments.DepartmentName

FROM Employees

FULL JOIN Departments

ON Employees.DepartmentID = Departments.DepartmentID;

```

5. **CROSS JOIN**:

- Ye dono tables ke **cartesian product** ko return karta hai, yani **har row** of the left table ko **har row** of the right table ke saath combine karta hai.

- Result mein bohot zyada rows aa sakti hain.

**Example:**

```sql

SELECT Employees.Name, Departments.DepartmentName

FROM Employees

CROSS JOIN Departments;

```

6. **SELF JOIN**:

   - Ye ek hi table ko **do alag alias** ke through join karta hai.

   - Ye typically tab use hota hai jab ek table mein relationship ho, jaise employees aur unke managers ke beech.

   Example:

   ```sql

   SELECT E.Name AS Employee, M.Name AS Manager

   FROM Employees E

   LEFT JOIN Employees M

   ON E.ManagerID = M.EmployeeID;

   ```

Summary:
   ```

| JOIN Type | Description | Example Query |

|---|---|---|

| **INNER JOIN** | Only rows with matching data in both tables. | `INNER JOIN` |

| **LEFT JOIN** | All rows from the left table, and matched rows from the right table. | `LEFT JOIN` |

| **RIGHT JOIN** | All rows from the right table, and matched rows from the left table. | `RIGHT JOIN` |

| **FULL JOIN** | All rows from both tables, with NULL where no match is found. | `FULL JOIN` |

| **CROSS JOIN** | Cartesian product of both tables (all combinations of rows). | `CROSS JOIN` |

| **SELF JOIN** | Joins a table with itself (typically using aliases). | `SELF JOIN` |

Key Takeaways:

- **INNER JOIN**: Only matching rows from both tables.

- **LEFT JOIN**: All rows from the left table, and matching from the right.

- **RIGHT JOIN**: All rows from the right table, and matching from the left.

- **FULL JOIN**: All rows from both tables, with NULL where no match.

- **CROSS JOIN**: All possible combinations of rows from both tables.

- **SELF JOIN**: A table joined with itself.

JOINs allow you to combine data from multiple tables in different ways, depending on your requirements.

Summary:

Left Table: The table mentioned first in the query.

Right Table: The table mentioned second in the query (after the JOIN).

UNION and **UNION ALL** are both used to combine the results of two or more **SELECT** queries, but they have a key difference:

1. UNION:

- **Removes duplicates**: **UNION** combines the results of multiple **SELECT** queries and automatically removes any duplicate rows.

- It ensures that each row in the result is **unique**.

- Slower than **UNION ALL** because it checks for duplicates.

Example:

```sql
SELECT Name FROM Employees
UNION
SELECT Name FROM Managers;
```

- In this example, if the same name appears in both the **Employees** and **Managers** tables, it will only show once in the result.

2. UNION ALL:

- **Does not remove duplicates**: **UNION ALL** combines the results of multiple **SELECT** queries, but it includes **all** rows, even if they are duplicates.

- It is **faster** than **UNION** because it does not need to check for duplicates.

Example:

```sql
SELECT Name FROM Employees
UNION ALL
SELECT Name FROM Managers;
```

- In this example, if the same name appears in both the **Employees** and **Managers** tables, it will appear **twice** in the result.

Key Differences:

| **Feature** | **UNION** | **UNION

ALL** |

|---|---|---|

| **Duplicates** | Removes duplicates | Includes duplicates |

| **Performance** | Slower (due to duplicate checking) | Faster (no duplicate checking) |

| **Use Case** | When you need unique rows | When you want all rows, including duplicates |

When to Use:

- Use **UNION** when you need the final result to have no duplicates.

- Use **UNION ALL** when you need all rows, including duplicates, and want better performance.

To fetch **unique records** from a table, you can use the **`DISTINCT`** keyword in SQL. This keyword ensures that the results contain only unique rows, eliminating any duplicates.

Syntax:

```sql
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Example:

Suppose we have an **Employees** table, and we want to get the unique **departments** in which employees work:

```sql
SELECT DISTINCT Department
FROM Employees;
```

- This will return a list of **unique departments** from

the **Employees** table, removing any duplicates.

For Multiple Columns:

You can also use **`DISTINCT`** on multiple columns if you want to get unique combinations of values from those columns.

For example, if you want to get unique combinations of **Department** and **Position**:

```sql
SELECT DISTINCT Department, Position
FROM Employees;
```

- This will return unique combinations of **Department** and **Position** without any duplicates.

Key Points:

- **`DISTINCT`** eliminates duplicate rows based on the columns you specify.

- It works on one or more columns in the **SELECT** query.

INNER JOIN vs **OUTER JOIN**:

1. INNER JOIN:

- **Kya hai**: Ye join sirf un rows ko return karta hai jo dono tables mein match karti hain.

- **Example**: Agar employee ko department ke saath match karna ho, to wo employee dikhega jo department mein ho.

2. OUTER JOIN:

- **Kya hai**: Ye join un rows ko bhi return karta hai jo ek table mein hoti hain, lekin doosre table mein match nahi karti. Agar match nahi hota to **NULL** dikhata hai.

- **Types of OUTER JOINS**:

  - **LEFT OUTER JOIN (LEFT JOIN)**: Left table ki saari

rows aur matching right table rows. Agar right table mein match nahi hota to **NULL**.

- **RIGHT OUTER JOIN (RIGHT JOIN)**: Right table ki saari rows aur matching left table rows. Agar left table mein match nahi hota to **NULL**.

- **FULL OUTER JOIN**: Dono tables ki saari rows ko dikhata hai, agar match nahi hota to **NULL**.

Example Queries:

```sql
-- INNER JOIN: Matching rows dikhata hai
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
INNER JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

```
-- LEFT JOIN: Left table ki saari rows dikhata hai, matching right table rows ke saath

SELECT Employees.Name,
Departments.DepartmentName

FROM Employees

LEFT JOIN Departments

ON Employees.DepartmentID =
Departments.DepartmentID;


-- RIGHT JOIN: Right table ki saari rows dikhata hai, matching left table rows ke saath

SELECT Employees.Name,
Departments.DepartmentName

FROM Employees

RIGHT JOIN Departments

ON Employees.DepartmentID =
Departments.DepartmentID;


-- FULL OUTER JOIN: Dono tables ki saari rows dikhata
```

hai, NULL agar match nahi ho

SELECT Employees.Name,
Departments.DepartmentName

FROM Employees

FULL OUTER JOIN Departments

ON Employees.DepartmentID =
Departments.DepartmentID;

```

### **Summary**:

- **INNER JOIN**: Sirf matching rows ko dikhata hai.

- **OUTER JOIN**: Non-matching rows ko bhi dikhata hai (NULL ke saath).

  - **LEFT OUTER JOIN**: Left table ki saari rows dikhata hai.

  - **RIGHT OUTER JOIN**: Right table ki saari rows dikhata hai.

  - **FULL OUTER JOIN**: Dono tables ki saari rows dikhata hai.
```

Difference Between Unique Key and Primary Key:

1. **Uniqueness**:

   - **Primary Key**: Ek table mein **sirf ek primary key** hoti hai, jo ki **uniqueness** guarantee karti hai. Isme **NULL** values allowed nahi hoti.

   - **Unique Key**: Unique key bhi **values ko unique** banati hai, lekin ek table mein **multiple unique keys** ho sakti hain. Isme **ek ya zyada NULL values** allow hoti hain (depending on the DBMS).

2. **NULL Values**:

   - **Primary Key**: Primary key me **NULL** nahi hoti.

   - **Unique Key**: Unique key me **ek ya zyada NULL values** allow hoti hain.

3. **Purpose**:

   - **Primary Key**: Data ko uniquely identify karne ke liye use hoti hai.

- **Unique Key**: Table ke data ko ensure karne ke liye use hoti hai ki kisi column mein duplicate values na ho, lekin primary key ki tarah required nahi hoti.

4. **Indexing**:

- **Primary Key**: Automatically **clustered index** create hota hai (agar explicitly nahi diya gaya ho).

- **Unique Key**: **Non-clustered index** automatically create hota hai.

Example:

```sql
-- Primary Key Example
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100)
);

-- Unique Key Example
```

```
CREATE TABLE Employees (
    EmployeeID INT UNIQUE,
    Email VARCHAR(100) UNIQUE,
    Name VARCHAR(100)
);
```

Summary:

- **Primary Key**: **Single**, **no NULLs**, unique identifier.

- **Unique Key**: **Multiple allowed**, **NULLs allowed**, ensures uniqueness but not an identifier.

1. How does GROUP BY work in SQL?

- **GROUP BY** ka use SQL mein rows ko ek ya zyada columns ke basis par group karne ke liye hota hai. Ye tab

use hota hai jab aapko aggregate functions (jaise SUM, COUNT, AVG) apply karni hoti hain ek group of rows par.

- **Example**:

```sql
SELECT Department, COUNT(*)
FROM Employees
GROUP BY Department;
```

- Is example mein, `GROUP BY` Employees ko unke department ke hisaab se group karega aur har department ka count dega.

---

2. What is the purpose of the ORDER BY clause?

- **ORDER BY** clause ka use SQL mein rows ko ascending (default) ya descending order mein sort karne ke liye hota hai. Aap specify karte hain kaunse columns

par sorting karni hai.

- **Example**:

```sql
SELECT Name, Salary
FROM Employees
ORDER BY Salary DESC;
```

- Is example mein, `ORDER BY` salary ko descending order mein sort karega.

---

3. What are aggregate functions in SQL? Give examples.

- **Aggregate Functions** SQL mein wo functions hain jo multiple rows ko ek single value mein convert karte hain.

- **Common Aggregate Functions**:

  - **COUNT()**: Rows ka count return karta hai.

- **SUM()**: Numeric values ka sum return karta hai.
- **AVG()**: Numeric values ka average return karta hai.
- **MIN()**: Minimum value return karta hai.
- **MAX()**: Maximum value return karta hai.

Examples:

```sql
SELECT COUNT(*) FROM Employees; -- Count of all employees
SELECT SUM(Salary) FROM Employees; -- Total salary of all employees
SELECT AVG(Salary) FROM Employees; -- Average salary of all employees
SELECT MIN(Salary) FROM Employees; -- Minimum salary
SELECT MAX(Salary) FROM Employees; -- Maximum salary
```

---

4. What is the difference between COUNT(*) and COUNT(column_name)?

- **COUNT(*)**: Sabhi rows ka count karta hai, chahe wo NULL ho ya non-NULL.
- **COUNT(column_name)**: **Sirf non-NULL values** ka count karta hai us particular column mein.

Example:

```sql
SELECT COUNT(*) FROM Employees; -- Count of all rows (including NULLs)
SELECT COUNT(Salary) FROM Employees; -- Count of non-NULL salary values
```

**5. What are subqueries? How are they different

from JOINs?**

- **Subqueries**: Ye ek query hoti hai jo doosri query ke andar hoti hai. Subqueries ko `SELECT`, `INSERT`, `UPDATE`, ya `DELETE` statement ke andar use kiya ja sakta hai.

  - **Example**:

  ```sql

  SELECT Name FROM Employees

  WHERE DepartmentID = (SELECT DepartmentID FROM Departments WHERE DepartmentName = 'Sales');

  ```

  - Is example mein, ek query department ka ID search kar rahi hai aur doosri query employee ka name fetch kar rahi hai.

- **JOINs**: JOINs ka use tables ko combine karne ke liye hota hai jab aapko ek hi query mein multiple tables ke data ko fetch karna ho.

  - **Example**:

```sql
SELECT Employees.Name,
Departments.DepartmentName
FROM Employees
JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

Difference:

- **Subqueries** ek query ke andar hoti hai aur doosri query se data fetch karte hain, jabki **JOINs** tables ko combine karne ke liye use hote hain. Subqueries ko multiple rows ya single value return karne ke liye use kiya ja sakta hai, lekin JOINs directly tables ko join karte hain.

Summary:

1. **GROUP BY**: Rows ko group karne ke liye, aggregate functions ko apply karne ke liye.

2. **ORDER BY**: Rows ko ascending ya descending order mein sort karne ke liye.

3. **Aggregate Functions**: Functions jo multiple rows ko ek value mein convert karte hain (COUNT, SUM, AVG, MIN, MAX).

4. **COUNT(*)**: Sabhi rows ka count, **COUNT(column_name)**: Non-NULL values ka count.

5. **Subqueries**: Queries jo doosri query ke andar hoti hain, **JOINs** tables ko combine karte hain.

What are Queries?

- **Query** SQL mein ek request hoti hai jo database se data retrieve karne, modify karne, insert karne, ya delete karne ke liye hoti hai.

- SQL queries ko likhne ka main purpose hota hai data ko retrieve ya manipulate karna according to the user's requirements.

Types of Queries:

1. **SELECT Query**: Data ko fetch karne ke liye use hota hai.

   - **Example**: `SELECT * FROM Employees;`

2. **INSERT Query**: Naya data insert karne ke liye use hota hai.

   - **Example**: `INSERT INTO Employees (Name, Salary) VALUES ('John Doe', 50000);`

3. **UPDATE Query**: Existing data ko modify karne ke liye use hota hai.

   - **Example**: `UPDATE Employees SET Salary = 55000 WHERE EmployeeID = 1;`

4. **DELETE Query**: Data ko delete karne ke liye use hota hai.

   - **Example**: `DELETE FROM Employees WHERE EmployeeID = 1;`

Summary:

- **Query** SQL mein ek instruction hoti hai jo data ko retrieve ya manipulate karne ke liye execute ki jati hai. Queries ka use data ko fetch karne, insert karne, update

karne, aur delete karne ke liye hota hai.