**SQL Material**

**What is Database?**

A **database** is a **structured collection of data stored electronically**. This data can be in different formats **like text, video, images**, etc.

**What is Database Management System (DBMS)?**

It is **software** that helps you **manage and organize data in a database**. It makes it easy to store, update, and retrieve data efficiently.

**What are the Types of Databases?**

There are several types of databases, and here are the main ones:

1. **Relational Database (RDBMS)**:
   * Stores data in tables with rows and columns.
   * Example: MySQL, PostgreSQL, Oracle.
2. **NoSQL Database**:
   * Stores unstructured or semi-structured data, like JSON, key-value pairs, or documents.
   * Example: MongoDB, Cassandra, CouchDB.
3. **Object-Oriented Database**:
   * Stores data as objects, like in programming.
   * Example: db4o, ObjectDB.
4. **Graph Database**:
   * Stores data in a graph format with nodes and relationships (useful for connected data).
   * Example: Neo4j.
5. **Cloud Database**:
   * A database that runs on cloud services, allowing for flexible and scalable storage.
   * Example: Amazon RDS, Google Cloud SQL.
6. **Hierarchical Database**:
   * Organizes data in a tree-like structure, with parent-child relationships.
   * Example: IBM Information Management System (IMS).
7. **Distributed Database**:
   * Data is stored across multiple physical locations, often on different servers.
   * Example: Apache Cassandra.

**What is SQL?**

**SQL (Structured Query Language)** is a programming language used to interact with databases. It helps you **store, retrieve, update, and manage data** in a database.

SQL is the **language** used to **communicate with the DBMS**.

In simple terms:

* **DBMS** is the system that handles the database.
* **SQL** is the language you use to tell the DBMS what to do with the data.

**Difference between SQL & NOSQL?**

|  |  |
| --- | --- |
| **SQL Databases** | **NoSQL Databases** |
| Relational database system (uses tables) | Non-relational or distributed system (different formats) |
| Data stored in rows and columns (like a spreadsheet) | Data stored as key-value pairs, documents, graphs, or wide-columns |
| Fixed structure: You must set up the data format (like columns and data types) before adding data | Flexible: You can add different types of data without setting a structure first |
| Uses SQL (Structured Query Language) | Doesn't require SQL, uses different methods depending on the database |
| Best for structured data | Best for unstructured or semi-structured data |
| Scales vertically (you need a bigger, more powerful server) | Scales horizontally (you can add more servers to handle more data) |
| Follows ACID Model (Atomicity, Consistency, Isolation, Durability) | Follows BASE (Basically Available, Soft state, Eventual consistency) Model |
| Examples: MySQL, PostgreSQL, SQL Server | Examples: MongoDB, Cassandra, CouchDB |

**What are the different types of SQL commands?**

**A diagram of a language

AI-generated content may be incorrect.**

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

These commands define the structure of the database, such as creating or modifying tables.

* **CREATE :** To create a new table or database.
* CREATE TABLE table\_name (
* column1 datatype,
* column2 datatype,
* ...
* );

**Example:**

* CREATE TABLE Employees (
* ID INT,
* Name VARCHAR(100),
* Salary DECIMAL(10, 2)
* );
* **ALTER :** Used to modify an existing table, such as adding or deleting columns.
* ALTER TABLE table\_name
* ADD column\_name datatype;
* *-- OR*
* ALTER TABLE table\_name
* DROP COLUMN column\_name;

**Example:**

* ALTER TABLE Employees
* ADD Department VARCHAR(50);
* **DROP :** Used to permanently delete an entire table, database, or object. Once deleted, the data cannot be recovered, and you cannot roll back the operation.
* DROP TABLE table\_name;

**Example:**

* DROP TABLE Employees;
* **TRUNCATE :** Used to remove all rows from a table but keep the table structure intact. Unlike **DROP**, the table remains available for future use, and it is faster than using **DELETE** without a WHERE clause.
* TRUNCATE TABLE table\_name;

**Example:**

* TRUNCATE TABLE Employees;

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

These commands manipulate the data in tables (inserting, updating, deleting).

* **INSERT :** Used to insert data into a table.
* INSERT INTO table\_name (column1, column2, ...)
* VALUES (value1, value2, ...);

**Example:**

* INSERT INTO Employees (ID, Name, Salary)
* VALUES (1, 'John Doe', 5000.00);
* **UPDATE :** Used to modify existing data in a table. If the **WHERE** clause is not specified, it will update all the records in the table, which could lead to unintentional updates.
* UPDATE table\_name
* SET column1 = value1, column2 = value2, ...
* WHERE condition;

**Example:**

* UPDATE Employees
* SET Salary = 5500.00
* WHERE ID = 1;
* **DELETE :** Used to delete specific records from a table. The **WHERE** clause is essential for selecting which rows should be deleted. If the **WHERE** clause is omitted, all rows in the table will be deleted.
* DELETE FROM table\_name
* WHERE condition;

**Example:**

* DELETE FROM Employees
* WHERE EmployeeID = 101;

**3. DQL (Data Query Language):**

This command is used to retrieve data from the database.

* SELECT : To query or fetch data from a table.
* SELECT column1, column2, ...
* FROM table\_name
* WHERE condition;

**Example:**

* SELECT Name, Salary
* FROM Employees
* WHERE Salary > 4000;

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

These commands deal with the permissions and access control in the database.

* GRANT **:** Used to give users access rights or privileges to the database.
* GRANT privilege ON object TO user;

**Example:**

* GRANT SELECT ON Employees TO User1;
* REVOKE **:** Used to remove user access rights or privileges.
* REVOKE privilege ON object FROM user;

**Example:**

* REVOKE SELECT ON Employees FROM User1;

**5. TCL (Transaction Control Language) :**

These commands deal with managing transactions within the database.

* COMMIT **:** Used to save all changes made in a transaction permanently.

**Syntax & Example:**

* COMMIT;
* ROLLBACK **:** Used to undo changes made in a transaction before a COMMIT.

**Syntax & Example:**

* ROLLBACK;
* SAVEPOINT **:** Used to set a point within a transaction to which you can later roll back.
* SAVEPOINT savepoint\_name;

**Example:**

* SAVEPOINT Save1;

**What is the difference between DELETE, TRUNCATE, and DROP?**

* **DELETE**: Removes specific rows from a table based on the WHERE condition. Can be rolled back (transaction-safe).
* **TRUNCATE**: Removes all rows from a table but keeps the table structure. Cannot be rolled back.
* **DROP**: Deletes the entire table, including its structure and data. Cannot be rolled back.

**What are the Datatypes in the SQL ?**

In SQL, data types define the type of data that can be stored in a column. that determines the kind of data it can store, such as integers, characters, dates, etc.

* **Numeric Data Types :**
  + **INT -** whole numbers - (e.g., 1, 100, -50).
  + **FLOAT -** floating-point number - (e.g., 1.23).
  + **DECIMAL -** precise numbers with decimal points - (e.g., DECIMAL(10, 2) for 10 digits, 2 decimal places).
  + **BIGINT -** large whole numbers.
  + **SMALLINT -** smaller whole numbers.
  + **TINYINT -** very small integers (1 byte).
* **Character/String Data Types :**
  + **CHAR(size) :** Stores fixed-length strings
  + **VARCHAR(size) :** Stores variable-length strings, up to n characters.
  + **TEXT :** Used to store large blocks of text.
  + **NVARCHAR(size) :** Stores variable-length Unicode string
* **Date and Time Data Types :**
* **DATE :** date in the format YYYY-MM-DD (e.g., 2025-03-12).
* **TIME :** time in the format HH:MI:SS (e.g., 13:45:30).
* **DATETIME :** both date and time in the format YYYY-MM-DD HH:MI:SS.
* **TIMESTAMP :** the current timestamp automatically
* **YEAR :** A Year in four-digit format 1901 – 2155
* **Boolean Data Type :**
* **BOOLEAN :** Stores TRUE or FALSE.
* **Special Data Type:**
* **ENUM :** A string object that can have **only one value** from a predefined list.
* CREATE TABLE Employees (
* Name VARCHAR(50),
* Gender ENUM('Male', 'Female', 'Other')
* );
* **SET :** A string object that can have **zero or more values** from a predefined list.
* CREATE TABLE Employees (
* Name VARCHAR(50),
* Skills SET('HTML', 'CSS', 'JavaScript', 'Python', 'Java')
* );

**What is the Difference between CHAR() and VARCHAR() ?**

* **CHAR :**
* It is a fixed-length data type.
* Always takes the full length of the defined size, even if the data is shorter.
* If you specify CHAR(**10**) and store 'Hi', the database will store it as **'Hi ' (with 8 spaces).**
* **VARCHAR:**
* It is a variable-length data type.
* Uses only as much space as the data needs, so it's more storage-efficient for variable-length data.
* If you specify VARCHAR(**10**) and store 'Hi', it will store just 'Hi' **without extra spaces.**

**What is a clause in SQL ?**

* A **clause** in SQL is like **a rule or condition** that you add to a query to tell the database exactly what you want to do with the data.
* Clauses help in **selecting, filtering, grouping, sorting, and joining data** in a table.
* Some Common Clauses :
* SELECT , WHERE , GROUP BY , HAVING,
* ORDER BY, JOIN, LIMIT, DISTINCT
* DISTINCT :

The **DISTINCT** clause in SQL is used to **remove duplicate values** from the result set. It **returns only unique** (different) values from a column or a combination of columns.

* SELECT DISTINCT column1, column2, ...
* FROM table\_name;
* SELECT DISTINCT city
* FROM Customers;
* WHERE :

The **WHERE** clause in SQL is used to **filter records in a query** based on a specific condition. It helps you **fetch only those rows** from the table that **meet the condition** you specify.

* SELECT \*
* FROM Customers
* WHERE city = 'New York';
* SELECT \*
* FROM Customers
* WHERE city = 'New York';
* ORDER BY :

The **ORDER BY** clause in SQL is used to **sort the result set** in either ascending (ASC) or descending (DESC) order based on one or more columns.

* SELECT column1, column2, ...
* FROM table\_name
* ORDER BY column1 [ASC|DESC];
* SELECT name, salary
* FROM Employees
* ORDER BY salary DESC;
* GROUP BY :

The **GROUP BY** clause in SQL is used to **group rows** that have **the same values in specified columns** into summary rows

It is often used with **aggregate functions** like COUNT(), SUM(), AVG(), etc., to perform calculations on **each group of data.**

* SELECT column1, aggregate\_function(column2)
* FROM table\_name
* GROUP BY column1;
* SELECT customer\_id, SUM(order\_amount)
* FROM Orders
* GROUP BY customer\_id;
* HAVING :

The **HAVING** clause in SQL is used to filter the result set of a **GROUP BY** query.

( **filters records after grouping)**

* SELECT column1, aggregate\_function(column2)
* FROM table\_name
* GROUP BY column1
* HAVING condition;
* SELECT customer\_id, COUNT(order\_id)
* FROM Orders
* GROUP BY customer\_id
* HAVING COUNT(order\_id) > 5;
* LIMIT :

The **LIMIT** clause in SQL is used to **restrict the number of rows returned in a query result**. It is useful when you only want to display a specific number of rows.

* SELECT column1, column2, ...
* FROM table\_name
* LIMIT number\_of\_rows;
* SELECT \*
* FROM Employees
* LIMIT 5;

**What is Difference Between Where and Having ?**

* **WHERE** is used to filter rows **before any grouping** is done, while **HAVING** is used to filter groups **after the GROUP BY clause.**
* WHERE **cannot be** used with **aggregate functions** like COUNT(), but HAVING can.

**What are the Types of SQL Operators?**

In SQL, **operators** are used **to perform operations on data** in a query. They help in filtering, comparing, or manipulating values in a database.

* **Arithmetic Operators**

These are used to perform mathematical operations.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition | SELECT 5 + 2; |
| - | Subtraction | SELECT 5 - 2; |
| \* | Multiplication | SELECT 5 \* 2; |
| / | Division | SELECT 5 / 2; |
| % | Modulus (remainder) | SELECT 5 % 2; |

* **Comparison Operators**

These are used to compare two values.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Equal to | SELECT \* FROM Users WHERE age = 25; |
| < > or != | Not equal to | SELECT \* FROM Users WHERE age != 25; |
| > | Greater than | SELECT \* FROM Users WHERE age > 25; |
| < | Less than | SELECT \* FROM Users WHERE age < 25; |
| >= | Greater than or equal to | SELECT \* FROM Users WHERE age >= 25; |
| <= | Less than or equal to | SELECT \* FROM Users WHERE age <= 25; |

* **Logical Operators**

These are used to combine multiple conditions in a query.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| AND | Returns true if both conditions are true | SELECT \* FROM Users WHERE age > 25 AND salary > 50000; |
| OR | Returns true if either condition is true | SELECT \* FROM Users WHERE age > 25 OR salary > 50000; |
| NOT | Reverses the result of a condition | SELECT \* FROM Users WHERE NOT age = 25; |

* **BETWEEN Operator**

Used to filter values within a **specific range.**

* SELECT \* FROM Employees WHERE salary BETWEEN 3000 AND 6000;
* **IN Operator**

Used to filter rows based on a **list of values**.

* SELECT \* FROM Employees WHERE department IN ('HR', 'Finance', 'Sales');
* **IS NULL Operator**

Checks for NULL (missing) values.

* SELECT \* FROM Employees WHERE address IS NULL;
* **EXISTS Operator**

Checks if a subquery returns any records.

* SELECT \* FROM Employees WHERE EXISTS (SELECT \* FROM Orders WHERE Employees.id = Orders.employee\_id);
* **LIKE Operator**

The **LIKE** operator is used in a **WHERE** clause to **search for a specified pattern** in a column.

The LIKE operator supports two wildcard characters :

* **% (Percentage sign)**: Represents zero, one, or multiple characters.
* **\_ (Underscore)**: Represents a single character.
* SELECT column1, column2, ...
* FROM table\_name
* WHERE column\_name LIKE pattern;
* **Starts With :** use the **%** wildcard **after** the characters.
* *-- Example: Find names that start with 'A'*
* SELECT \* FROM employees
* WHERE name LIKE 'A%';
* **Ends With :** use the **%** wildcard **before** the characters.
* *-- Example: Find names that end with 'n'*
* SELECT \* FROM employees
* WHERE name LIKE '%n';
* **Contains :** use **%** **before** and **after** the characters.
* *-- Example: Find names that contain 'son'*
* SELECT \* FROM employees
* WHERE name LIKE '%son%';
* **At Least n Characters :** To filter rows where a column has **at least a specific number of characters**, you can use underscores ( \_ ) to represent each character.
* *-- Example: Find names that are at least 5 characters long*
* SELECT \* FROM employees
* WHERE name LIKE '\_\_\_\_\_%' ;
* **Match Vowels :** you can use the **[ ]** to match a set of characters (i.e., the vowels)

**MYSQL not Supported this type [ ]**

* *-- Example: Find names that start with a vowel (A, E, I, O, U)*
* SELECT \* FROM employees
* WHERE name LIKE '[AEIOU]%';

You can use **REGEXP** to check if a name starts with a vowel

* *-- Example: Find names that start with a vowel (A, E, I, O, U) in MySQL*
* SELECT \* FROM employees
* WHERE name REGEXP '^[AEIOU]'; ‘[AEIOU]$’ – End with Vowels
* **Match Consonants :**
* SELECT \* FROM employees
* WHERE NOT name REGEXP '^[aeiouAEIOU]'; or '^[bcdfghjklmnpqrstvwxyz]'
* **Match First and Last With Vowel:**
* SELECT DISTINCT column\_name
* FROM table\_name
* WHERE column\_name REGEXP '^[aeiouAEIOU].\*[aeiouAEIOU]$';

**ANY Operator :**

* The **ANY** operator is used to **compare a value to any value in a list** or subquery.
* It returns **TRUE** if **at least one value from the set** satisfies the condition.
* SELECT column\_name(
* FROM table\_name
* WHERE column\_name comparison\_operator ANY (subquery);
* SELECT name, salary
* FROM employees
* WHERE salary > ANY (SELECT salary FROM employees WHERE department\_id = 2);

**ALL Operator:**

* The ALL operator is used to compare **a value to all values in a list or subquery.**
* It returns TRUE only if the condition is satisfied by **all** values in the set.
* SELECT column\_name(s)
* FROM table\_name
* WHERE column\_name comparison\_operator ALL (subquery);
* SELECT name, salary
* FROM employees
* WHERE salary > ALL (SELECT salary FROM employees WHERE department\_id = 2);

**What is REGEX Patterns In MYSQL ?**

**Regex in MySQL** refers to the use of **regular expressions** for pattern matching in SQL queries.

### **Common Features of MySQL Regex:**

1. **^ :** Matches the beginning of a string.
   * + Example: ^a matches any string starting with 'a'
2. **$ :** Matches the end of a string.
   * + Example: z$ matches any string ending with 'z'.
3. **. :** Matches any single character.
   * + Example: a.b matches 'aab', 'acb', 'a1b', etc.
4. **[abc] :** Matches any character within the brackets.
   * + Example: [aeiou] matches any vowel.
5. **[^abc] :** Matches any character **not** within the brackets.
   * + Example: [^aeiou] matches any character that is not a vowel.
6. **| :**  Acts as a logical OR between patterns.
   * + Example: cat|dog matches 'cat' or 'dog'.
7. **\* :** Matches 0 or more occurrences of the preceding character or group.
   * + Example: a\* matches 'a', 'aa', '', etc.
8. **+ :** Matches 1 or more occurrences of the preceding character or group.
   * + Example: a+ matches 'a', 'aa', but not '' (empty).
9. **? :** Matches 0 or 1 occurrence of the preceding character or group.
   * + Example: a? matches either 'a' or ''.
10. **{n}** : Matches exactly n occurrences of the preceding character or group.
    * + Example: a{3} matches 'aaa'.

**Find Names that Contain Only Alphabets**

SELECT name

FROM employees

WHERE name REGEXP '^[A-Za-z]+$';

**Vowel in the Second Position**

SELECT name

FROM employees

WHERE name REGEXP '^.[aeiouAEIOU]';

**Phone Number Matches the Format (123-456-7890)**

SELECT phone\_number

FROM employees

WHERE phone\_number REGEXP '^[0-9]{3}-[0-9]{3}-[0-9]{4}$';

**What is Aggregate functions in SQL?**

**Aggregate functions** in SQL are used to perform **calculations on multiple rows** of a table and return a **single result**. They are often used with the **GROUP BY** clause to perform operations on groups of data. Aggregate functions **ignore NULL values**.

* **Count() :** Counts the number of rows that match a specified condition.
* SELECT COUNT(employee\_id)
* FROM employees
* WHERE department = 'Sales';
* **SUM() :** Adds up the values in a column (numeric).
* SELECT SUM(salary)
* FROM employees;
* **AVG() :** Calculates the average value of a numeric column.
* SELECT AVG(salary)
* FROM employees
* WHERE department = 'IT';
* **MIN() :** Returns the smallest value in a column.
* SELECT MIN(salary)
* FROM employees;
* **MAX() :** Returns the largest value in a column.
* SELECT MAX(salary)
* FROM employees;

**What is Aliases in SQL?**

**Aliases** in SQL are used to give a **temporary name** to a table or a column in a query. The **AS** keyword is **optional**, but using it improves readability.

SELECT column\_name AS alias\_name

FROM table\_name;

SELECT first\_name AS 'First Name', last\_name AS 'Last Name'

FROM employees;

**Use quotes (single or double)**:

* When the alias has spaces (e.g., 'Full Name').
* When the alias has special characters (e.g., 'emp#1').
* When the alias is a reserved word (e.g., `select`).

**No quotes needed**:

* For single-word aliases without spaces or special characters (e.g., monthly\_income).

**What is Join in SQL and its types ?**

In SQL, **joins** are used to **combine rows from two or more tables** based on a related column between them.

* INNER JOIN **:**
* Returns records that have **matching values** in both tables.
* if you write just **JOIN** without specifying the type, it defaults to **INNER JOIN**.



* SELECT columns
* FROM table1
* INNER JOIN table2
* ON table1.common\_column = table2.common\_column;
* LEFT JOIN **(**LEFT OUTER JOIN**) :**

* Returns **all records from the left** table and the **matched records from the right** table.
* If there is **no match**, **NULL values** are returned for columns from the **right table.**



* SELECT columns
* FROM table1
* LEFT JOIN table2
* ON table1.common\_column = table2.common\_column;
* RIGHT JOIN (RIGHT OUTER JOIN) :
* Returns **all records from the right** table and the **matched records from the left** table.
* If there is **no match, NULL** values are returned for columns from the **left table.**



* SELECT columns
* FROM table1
* RIGHT JOIN table2
* ON table1.common\_column = table2.common\_column;
* FULL JOIN (FULL OUTER JOIN) :
* Returns **all records** when **there is a match in either the left or right table**.
* If there is no match, NULL values are returned for columns from the table with no match.



* SELECT columns
* FROM table1
* FULL OUTER JOIN table2
* ON table1.common\_column = table2.common\_column;
* SELF JOIN :
* A self join is a regular join, but the table is joined with itself.
* SELECT a.column\_name, b.column\_name
* FROM table\_name a, table\_name b
* WHERE condition;
* CROSS JOIN :
* Returns the Cartesian product of the two tables. In other words, **it returns all combinations of rows from the two tables.**
* SELECT columns
* FROM table1
* CROSS JOIN table2;

**What is UNION and Union All Operator?**

* **UNION :**

* **Combines** the result sets of two or more SELECT queries, **removing duplicates**.
* Returns only **unique rows** (no duplicates)
* SELECT column\_name(s) FROM table1
* UNION
* SELECT column\_name(s) FROM table2;
* **UNION ALL :**
* Combines the result sets of two or more SELECT queries, **including duplicates**.
* SELECT column\_name(s) FROM table1
* UNION ALL
* SELECT column\_name(s) FROM table2;

**How to Copy rows from one table to another ?**

* **INSERT INTO** statement in combination with a **SELECT** query is used to **copy rows** from **one table** to **another** in MySQL

INSERT INTO target\_table (column1, column2, column3, ...)

SELECT column1, column2, column3, ...

FROM source\_table

WHERE conditions;

INSERT INTO employees\_backup (id, name, position, salary)

SELECT id, name, position, salary

FROM employees;

**What is CASE Expression in SQL ?**

* The **CASE** expression is a **conditional expression** that allows you to add **if-else-like logic** to your SQL queries.
* It evaluates conditions in a **sequential order** and returns a specific result based on the first condition that is met.
* There are two types of CASE expressions: **Simple CASE** and **Searched CASE**.

**Simple CASE : Compares** an expression to a **list of possible values** and returns the corresponding result.

CASE expression

WHEN value1 THEN result1

WHEN value2 THEN result2

...

ELSE default\_result

END

SELECT name,

CASE department

WHEN 'HR' THEN 'Human Resources'

WHEN 'IT' THEN 'Information Technology'

WHEN 'SALES' THEN 'Sales Department'

ELSE 'Unknown Department'

END AS department\_name

FROM employees;

**Searched CASE** : It allows you to **test multiple conditions** (instead of comparing a single value) and returns the **first THEN result** for which the **condition is true.**

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

...

ELSE default\_result

END

SELECT name, salary,

CASE

WHEN salary >= 100000 THEN 'High Salary'

WHEN salary >= 50000 THEN 'Medium Salary'

ELSE 'Low Salary'

END AS salary\_range

FROM employees;

* If no conditions are TRUE, the ELSE clause is executed. If there is **no ELSE** clause and **no condition is met**, NULL is returned.

**What is Constraint in SQL ?**

* SQL constraints are **rules applied to columns** or tables in a relational **database** to **limit** the type of data that can be inserted, updated, or deleted.
* Constraints are used to **prevent invalid data** from being inserted into the database
* Constraints can be **specified** when the table is created with the **CREATE TABLE** statement, or **after the table** is created with the **ALTER TABLE** statement.

**Types of Constraints** :

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT

NOT NULL :

* Ensures that a **column cannot have a NULL value**. Every row must have a value for that column.
* CREATE TABLE employees (
* employee\_id INT,
* name VARCHAR(100) NOT NULL
* );

UNIQUE :

* Ensures that all values in a column (or combination of columns) are **unique** **across all rows in the table.**
* CREATE TABLE employees (
* employee\_id INT UNIQUE,
* email VARCHAR(255) UNIQUE
* );

PRIMARY KEY :

* A **Primary Key** is a column (or a set of columns) in a table that uniquely identifies each row
* A **PRIMARY KEY** combines the **UNIQUE** and **NOT NULL** constraints
* A **table** can have **only one primary key**, but it can consist of **multiple columns** (composite primary key).
* CREATE TABLE employees (
* employee\_id INT PRIMARY KEY,
* name VARCHAR(100)
* );
* ALTER TABLE Persons
* ADD PRIMARY KEY (ID);

FOREIGN KEY **:**

* A **Foreign Key** is a column or a set of columns that establishes a **relationship between two tables.**
* It **refers** to the **Primary Key of another table** and enforces referential integrity
* The table with the **foreign key** is called the **child table**, and the table with the **primary key** is called the **referenced or parent table**.
* CREATE TABLE orders (
* order\_id INT PRIMARY KEY,
* employee\_id INT,
* FOREIGN KEY (employee\_id) REFERENCES employees(employee\_id)
* );
* ALTER TABLE Orders
* ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

CHECK **:**

* Ensures that all values in a column meet a **specific condition**. It **checks that the values** being inserted or updated satisfy the defined condition.
* CREATE TABLE employees (
* employee\_id INT,
* salary DECIMAL(10, 2),
* CHECK (salary > 0)
* );

DEFAULT **:**

* Provides a **default value** for a column if **no value is specified** during insertion
* CREATE TABLE employees (
* employee\_id INT,
* start\_date DATE DEFAULT CURRENT\_DATE
* );

**What is AUTO\_INCREMENT in SQL ?**

An AUTO\_INCREMENT field in SQL (specifically in MySQL and similar databases) is a column that **automatically generates a unique number whenever a new record is inserted into the table**.

CREATE TABLE employees (

employee\_id INT AUTO\_INCREMENT,

name VARCHAR(100),

department VARCHAR(100),

PRIMARY KEY (employee\_id)

);

**SQL DATES**

* In MySQL, dates are handled using several date and time data types, as well as built-in functions to work with them. MySQL provides various ways to store, format, and manipulate dates.

**MySQL Date and Time Data Types:**

* **DATE**: Stores the date in the format YYYY-MM-DD.

Example: '2025-03-14'

* **DATETIME**: Stores both the date and time in the format YYYY-MM-DD HH:MM:SS.

Example: '2025-03-14 13:45:30'

* **TIMESTAMP**: Similar to DATETIME, but this value is stored as the number of seconds since the Unix Epoch (1970-01-01 00:00:00 UTC). It changes according to the timezone settings.

Example: '2025-03-14 13:45:30'

* **TIME**: Stores the time in the format HH:MM:SS.

Example: '13:45:30'

* **YEAR**: Stores a year in the format YYYY.

Example: '2025'

**Common MySQL Date Functions :**

* **CURRENT\_DATE()** and **CURDATE() :** Returns the current date in YYYY-MM-DD format.

SELECT CURRENT\_DATE(); *-- Example: '2025-03-14'*

SELECT CURDATE(); *-- Example: '2025-03-14*

* **CURRENT\_TIME()** and **CURTIME() :** Returns the current time in HH:MM:SS format.

SELECT CURRENT\_TIME(); *-- Example: '13:45:30'*

SELECT CURTIME(); *-- Example: '13:45:30'*

* **NOW() :** Returns both the current date and time in YYYY-MM-DD HH:MM:SS format.
* SELECT NOW(); *-- Example: '2025-03-14 13:45:30'*
* **DATE\_ADD()** : Adds a specific interval to a date.

**Syntax:**

**DATE\_ADD(date, INTERVAL value unit)**

**Units** : Day , Month, Year, Hour, Minute, Second

**Add Days to a Date :**

SELECT DATE\_ADD('2025-03-14', INTERVAL 5 DAY); *-- Example: '2025-03-19'*

* **DATE\_SUB()** : Subtracts a specific interval from a date.

**Subtract Days to a Date**

SELECT DATE\_SUB('2025-03-14', INTERVAL 10 DAY); *-- Example: '2025-03-04'*

* **DATEDIFF() :** Returns the difference in days between two dates.

SELECT DATEDIFF('2025-03-20', '2025-03-14'); *-- Example: 6*

* **EXTRACT()** : Extracts a specific part of a date, such as year, month, or day.

**Syntax :**

**EXTRACT(unit FROM date)**

**Units** : Day , Month, Year, Hour, Minute, Second

**Extract Year**

SELECT EXTRACT(YEAR FROM '2025-03-14'); -- 2025

**Extract Hour**

SELECT EXTRACT(HOUR FROM '2025-03-14 08:45:00'); -- 08

* **STR\_TO\_DATE()** : Converts a string into a date.

**Syntax :**

**STR\_TO\_DATE(string, format)**

SELECT STR\_TO\_DATE('14-03-2025', '%d-%m-%Y'); *-- Example: '2025-03-14'*

**Common Format Specifiers:**

* %Y - Year (4 digits)
* %m - Month (2 digits)
* %d - Day (2 digits)
* %H - Hour (2 digits, 24-hour format)
* %i - Minutes (2 digits)
* %s - Seconds (2 digits)
* %p - AM/PM
* **DATE\_FORMAT()** : Formats a date into a specified format.

Syntax :

**DATE\_FORMAT(date, format)**

**Common Format Specifiers:**

A screenshot of a phone

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.

SELECT DATE\_FORMAT(NOW(), '%W, %M %d, %Y'); *-- Example: 'Friday, March 14, 2025'*

* **LAST\_DAY()** : Returns the last day of the month for a given date.

SELECT LAST\_DAY('2025-03-14'); *-- Example: '2025-03-31'*

* **+ INTERVAL** :

SELECT date\_column + INTERVAL n unit AS result\_column

FROM table\_name;

* **- INTERVAL :**

SELECT date\_column - INTERVAL n unit AS result\_column

FROM table\_name;

* **DATE() :** Extracts the date part from a date or datetime expression

SELECT DATE(NOW());

* **TIME()** : Extracts the time portion from a datetime value.

SELECT TIME(NOW());

* **YEAR() :** Extracts the year from a date.