**What is SQL?**

SQL (Structured Query Language) is a standard Database language designed for managing and manipulating relational databases. It allows users to create, retrieve, update, and delete data within a database. SQL is highly versatile and used across different database systems, including MySQL, PostgreSQL, Oracle, SQL Server, and SQLite.

SQL works with tables, where data is stored in rows and columns. Its syntax is simple and English-like, making it easy to understand and use.

### What is a Database?

A **database** is an organized collection of structured information or data, typically stored electronically in a computer system. It enables efficient data management, retrieval, and manipulation. Databases are designed to handle large amounts of data while ensuring data integrity and security.

**Purpose of SQL**

The primary purpose of SQL is to interact with databases to perform various operations like:

1. **Data Querying**: Retrieve specific data using SELECT statements.
2. **Data Manipulation**: Insert, update, delete, and alter data in a database.
3. **Data Definition**: Create or modify the structure of database objects such as tables, indexes, and views.
4. **Access Control**: Manage access permissions and ensure data security using SQL commands.
5. **Transaction Control**: Manage transactions and ensure the integrity of data during multiple operations.

**Who Developed SQL and When Was It Developed?**

SQL (Structured Query Language) was developed in the early 1970s by **IBM** researchers **Donald D. Chamberlin** and **Raymond F. Boyce**. They were part of the team working on IBM’s pioneering database project known as **System R**, which was designed to implement a relational database management system (RDBMS).

SQL was initially called **SEQUEL** (Structured English Query Language), which was inspired by the **relational model of data** proposed by **Edgar F. Codd** in 1970. Codd’s work on the relational model revolutionized the way databases were designed and managed, enabling the structured organization of data using tables and relationships. SEQUEL was later renamed SQL due to trademark issues.

**Timeline of SQL Development:**

* **1970**: Edgar F. Codd publishes a paper introducing the relational model of data.
* **1973–1974**: IBM researchers Chamberlin and Boyce develop SEQUEL, which was later renamed SQL.
* **1979**: Oracle Corporation (then known as Relational Software Inc.) releases the first commercial SQL-based RDBMS, Oracle V2.
* **1986**: SQL becomes a standard when the **American National Standards Institute (ANSI)** adopts SQL as the standard relational database query language.
* **1987**: SQL is adopted as an international standard by the **International Organization for Standardization (ISO)**.

**Who Should Learn SQL?**

* **Developers**: Both backend and full-stack developers need SQL to interact with databases in web applications.
* **Data Analysts**: SQL is essential for querying data and extracting insights for business intelligence.
* **Database Administrators**: They need SQL to manage databases, ensure data integrity, and optimize performance.
* **Data Scientists**: SQL is often used to clean and prepare large datasets for analysis.
* **System Administrators**: For managing database servers and tuning performance.

In general, anyone who works with data or systems involving relational databases will benefit from learning SQL.

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

A **Database Management System (DBMS)** is software that allows users to **create, manage, and interact with databases**. It provides a systematic and organized way to store, retrieve, update, and manage data. A DBMS ensures that the data is consistently organized and remains easily accessible.

The DBMS acts as an intermediary between the user and the database, ensuring that the data is stored safely, can be retrieved efficiently, and can be manipulated as needed by various applications. It also provides security, data integrity, and backup/recovery features.

#### **Key Functions of a DBMS:**

1. **Data Definition**: Helps define the structure of the data (schema) and the relationships between different data entities.
2. **Data Manipulation**: Allows users to query, update, and delete data from the database.
3. **Data Security**: Ensures that only authorized users can access or modify the database.
4. **Backup and Recovery**: Ensures data is safe from accidental loss or system failures, and provides a way to restore it.
5. **Concurrency Control**: Manages simultaneous data access to ensure that multiple users can interact with the database without conflicts.
6. **Data Integrity**: Ensures that the data remains accurate and consistent throughout its lifecycle.

### ****Types of DBMS****

DBMS can be categorized into various types based on the data models they use and the architecture they follow:

#### 1. **Hierarchical DBMS**

A **Hierarchical DBMS** organizes data in a **tree-like structure**, where each record has a single parent but can have multiple children. This model is good for representing hierarchical relationships like an organizational structure or a file system.

* **Example**: IBM's Information Management System (IMS).

**Advantages**:

* Simple and fast for hierarchical data.
* Efficient for one-to-many relationships.

**Disadvantages**:

* Limited flexibility (difficult to restructure or extend the hierarchy).
* Requires knowledge of the hierarchical path to access the data.

#### 2. **Network DBMS**

A **Network DBMS** organizes data in a **graph structure**, allowing each record to have multiple parent and child records (many-to-many relationships). This model is more flexible than the hierarchical model.

* **Example**: Integrated Data Store (IDS).

**Advantages**:

* Efficient for complex relationships.
* Supports many-to-many relationships.

**Disadvantages**:

* Complex structure makes it difficult to manage.
* Requires specialized knowledge for querying.

#### 3. **Relational DBMS (RDBMS)**

A **Relational DBMS** organizes data into **tables (relations)** that are made up of rows and columns. It uses **SQL (Structured Query Language)** to manage and query data. Data in relational databases is highly structured, and relationships between tables can be created using foreign keys.

* **Examples**: MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

**Advantages**:

* Simple and intuitive structure (tables).
* Supports powerful querying using SQL.
* Enforces data integrity with primary and foreign keys.
* ACID compliance ensures data reliability.

**Disadvantages**:

* Performance can be affected when scaling up with large datasets.
* Not ideal for unstructured data (like documents, images, etc.).

#### 4. **Object-Oriented DBMS (OODBMS)**

An **Object-Oriented DBMS** stores data in the form of **objects**, similar to how data is handled in object-oriented programming languages like Java or C++. Each object can contain data (attributes) and methods (operations).

* **Examples**: ObjectDB, db4o.

**Advantages**:

* Suitable for applications that use complex data structures (e.g., multimedia, engineering, etc.).
* Better integration with object-oriented programming languages.

**Disadvantages**:

* Slower performance compared to RDBMS for simple queries.
* Less widespread than relational databases, resulting in fewer tools and support.

#### 5. **Document-Oriented DBMS (NoSQL)**

A **Document-Oriented DBMS** stores, retrieves, and manages data as **documents**, usually in formats like **JSON** or **XML**. It is a type of **NoSQL** database that is highly flexible and can handle unstructured or semi-structured data.

* **Examples**: MongoDB, CouchDB.

**Advantages**:

* Schema-less design, which allows for flexible data models.
* Handles unstructured data well (e.g., JSON documents).
* Scales easily horizontally across distributed servers.

**Disadvantages**:

* Lacks the strict consistency and ACID properties of RDBMS.
* Less suited for applications that need structured data and complex relationships.

#### 6. **Key-Value Stores (NoSQL)**

A **Key-Value Store DBMS** is a **simple NoSQL database** that stores data as **key-value pairs**. The key is used as a unique identifier, and the value can be any type of data.

* **Examples**: Redis, Amazon DynamoDB, Riak.

**Advantages**:

* Fast and simple.
* Highly scalable for handling large volumes of simple data.

**Disadvantages**:

* Limited querying capabilities (compared to SQL databases).
* Not suitable for complex data relationships.

#### 7. **Column-Oriented DBMS (NoSQL)**

A **Column-Oriented DBMS** stores data in columns rather than rows. This model is efficient for handling large amounts of data, especially for analytical queries where specific columns are queried frequently.

* **Examples**: Apache Cassandra, HBase.

**Advantages**:

* Efficient for read-heavy operations and analytics.
* Scales horizontally across distributed clusters.

**Disadvantages**:

* Less efficient for transactional data and writes.
* Complex to set up and maintain.

#### 8. **Graph DBMS**

A **Graph DBMS** stores data in the form of **nodes, edges, and properties**, making it ideal for representing relationships and connections. It is useful in applications where data is highly interconnected, such as social networks.

* **Examples**: Neo4j, Amazon Neptune.

**Advantages**:

* Excellent for handling complex relationships.
* Allows for fast querying of paths and connections in large datasets.

**Disadvantages**:

* Not suitable for simple, tabular data.
* Performance can degrade with a large number of nodes or connections.

**What Are the Subsets of SQL?**

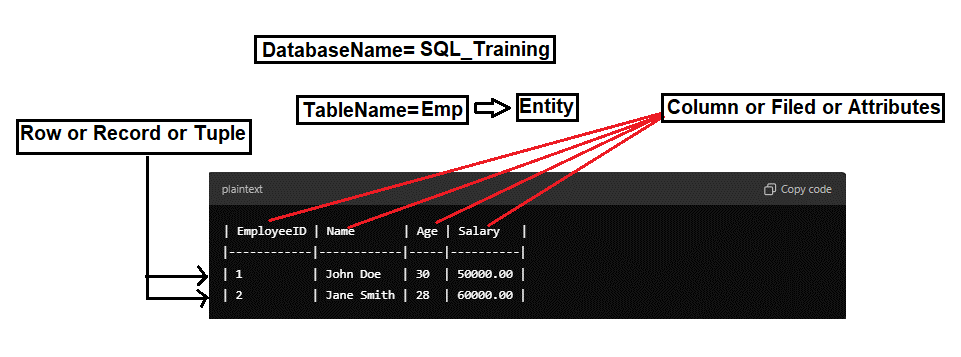
SQL can be categorized into different subsets based on the type of operations performed:

1. **Data Definition Language (DDL)**: Defines the structure and schema of a database.
2. **Data Manipulation Language (DML)**: Deals with data manipulation within tables.
3. **Data Control Language (DCL)**: Manages access control to the database.
4. **Transaction Control Language (TCL)**: Controls the execution of transactions to maintain data integrity.
5. **Data Query Language(DQL)**: Used to fetch data from database

**Database Tables**

A **table** is a collection of related data entries in a database. It consists of:

* **Rows (Records, Tuple)**: Each row represents a single data entry or instance of the data structure.
* **Columns (Fields, Attributes)**: Each column represents a specific attribute of the data (e.g., name, age, salary).



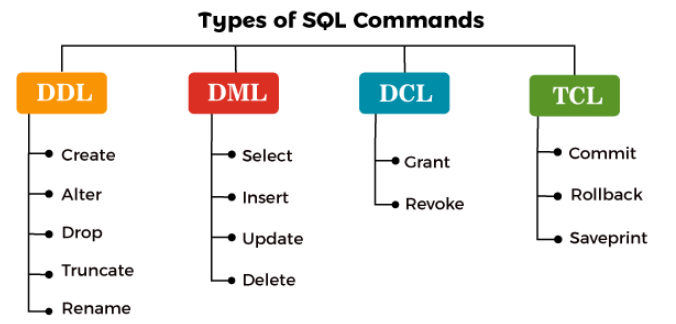
**Table Records**

A **record** (or row) in a table is a single, structured data entry that consists of values for each of the table’s columns. Each record is unique and identifiable, often through a primary key, which is a unique identifier for each record.

**Example of Records**:

* The first record: 1, John Doe, 30, 50000.00
* The second record: 2, Jane Smith, 28, 60000.00

Subset of SQL:



**Data Definition Language (DDL)**

DDL is used to define or modify the structure of database objects like tables, indexes, and views. Key commands include:

* **CREATE**: Creates a new table, view, or database.

CREATE TABLE Students (

ID INT PRIMARY KEY,

Name VARCHAR(100),

Age INT

);

* **ALTER**: Modifies an existing table or database object (e.g., adding/removing columns).

ALTER TABLE Students ADD Email VARCHAR(100);

* **DROP**: Deletes an existing table, database, or index.

DROP TABLE Students;

* **TRUNCATE**: Removes all rows from a table but keeps its structure.

TRUNCATE TABLE Students;

**Data Query Language(DQL)**

DQL Allows you to fetch data from database

* **SELECT**: Retrieves data from a table.

SELECT \* FROM Students;

**Data Manipulation Language (DML)**

DML allows you to manipulate data within tables. It includes commands such as:

* **INSERT**: Adds new rows to a table.

INSERT INTO Students (ID, Name, Age) VALUES (1, 'John', 20);

* **UPDATE**: Modifies existing data in a table.

UPDATE Students SET Age = 21 WHERE ID = 1;

* **DELETE**: Removes rows from a table.

DELETE FROM Students WHERE ID = 1;

**Data Control Language (DCL)**

DCL manages user access and permissions in a database. Common DCL commands are:

* **GRANT**: Provides specific privileges to users (e.g., SELECT, INSERT, UPDATE).

GRANT SELECT ON Students TO user1;

* **REVOKE**: Removes privileges from users.

REVOKE SELECT ON Students FROM user1;

### ****TCL (Transaction Control Language) in SQL****

**TCL** commands are used to manage **transactions** in a database. A transaction is a sequence of one or more SQL operations that are executed as a single unit of work. Transactions ensure **data integrity** by grouping operations, making it possible to commit or roll back multiple changes simultaneously.

TCL commands are crucial for managing transactions to ensure **ACID** properties (Atomicity, Consistency, Isolation, and Durability).

### ****Key TCL Commands****:

1. **COMMIT**
2. **ROLLBACK**
3. **SAVEPOINT**

### ****1. COMMIT****

The COMMIT command is used to **permanently save** all the changes made in a transaction. Once a transaction is committed, the changes cannot be undone by a ROLLBACK.

#### **Syntax**:

COMMIT;

#### **Example**:

BEGIN TRANSACTION;

UPDATE Employees SET Salary = 5000 WHERE EmpID = 1;

COMMIT;

In this example, the salary of the employee with EmpID = 1 is updated, and the changes are saved permanently using the COMMIT command.

### ****2. ROLLBACK****

The ROLLBACK command is used to **undo** all the changes made in a transaction before it has been committed. This ensures that if something goes wrong, the database can be restored to its previous state.

#### **Syntax**:

ROLLBACK;

#### **Example**:

BEGIN TRANSACTION;

UPDATE Employees SET Salary = 5000 WHERE EmpID = 1;

ROLLBACK;

In this example, the ROLLBACK command is used to undo the salary update, so no changes are saved to the database.

### ****3. SAVEPOINT****

The SAVEPOINT command sets a **point within a transaction** to which you can later roll back. It allows for partial rollbacks in a long transaction, giving more control over how and where to undo changes.

#### **Syntax**:

SAVEPOINT savepoint\_name;

#### **Example**:

BEGIN TRANSACTION;

UPDATE Employees SET Salary = 5000 WHERE EmpID = 1;

SAVEPOINT sp1;

UPDATE Employees SET Salary = 6000 WHERE EmpID = 2;

ROLLBACK TO sp1;

COMMIT;

Here, a SAVEPOINT named sp1 is created. If something goes wrong after the savepoint, the ROLLBACK TO sp1 undoes changes made after sp1, but keeps the updates before it. In this case, the salary change for EmpID = 1 is saved, while the change for EmpID = 2 is rolled back.

**SQL vs. NoSQL**

SQL and NoSQL are two different types of database systems. While SQL databases are relational, NoSQL databases are designed for non-relational data models. Here’s a comparison:

| **Feature** | **SQL (Relational)** | **NoSQL (Non-Relational)** |
| --- | --- | --- |
| **Data Structure** | Uses tables (rows and columns). | Flexible data models: key-value pairs, documents, graphs, or wide-column stores. |
| **Schema** | Schema-based, with a fixed structure. | Schema-less, data can be unstructured or semi-structured. |
| **Query Language** | SQL (Structured Query Language). | Varies by system (e.g., MongoDB uses queries similar to JSON). |
| **Transactions** | ACID-compliant (ensures data integrity). | May not be fully ACID-compliant; instead focuses on eventual consistency. |
| **Scaling** | Vertical scaling (increasing power of the same server). | Horizontal scaling (adding more servers to distribute load). |
| **Use Cases** | Best for structured data and complex queries. | Ideal for unstructured data, high throughput, and real-time web apps. |

**SQL** is best suited for traditional applications like banking systems, HR software, and e-commerce sites where data consistency and relational operations are critical.

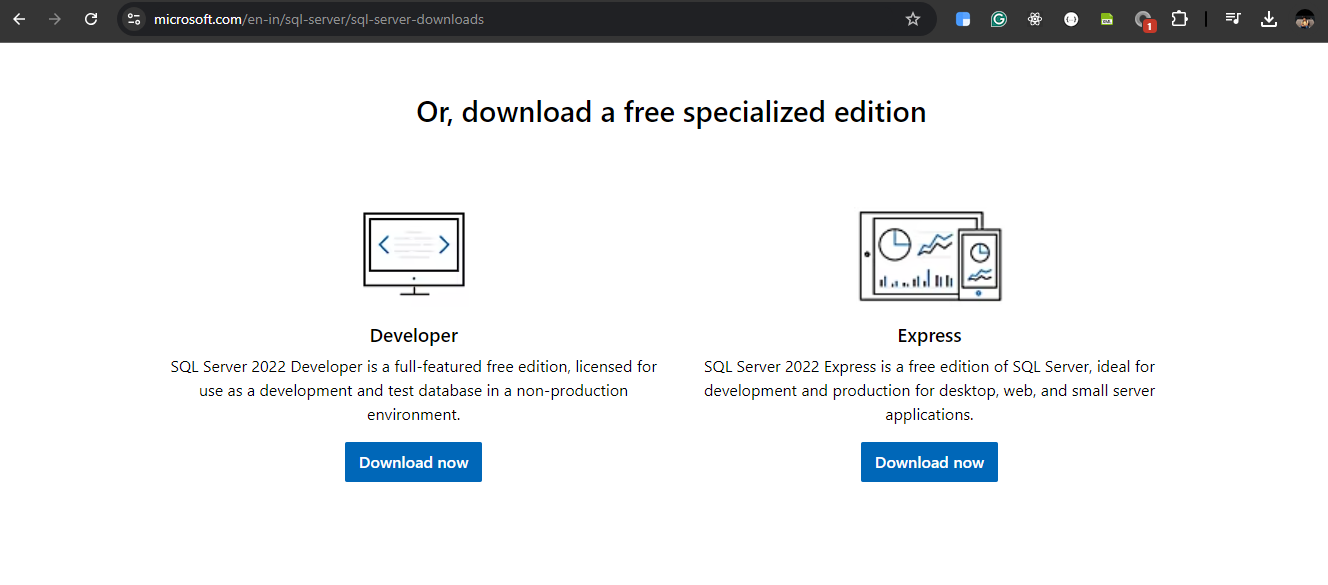
**NoSQL** is ideal for applications with large amounts of unstructured or semi-structured data, such as social media, real-time analytics, and IoT applications. Examples of NoSQL databases include MongoDB, Cassandra, and Redis.

Steps to download & install SQL Server database:

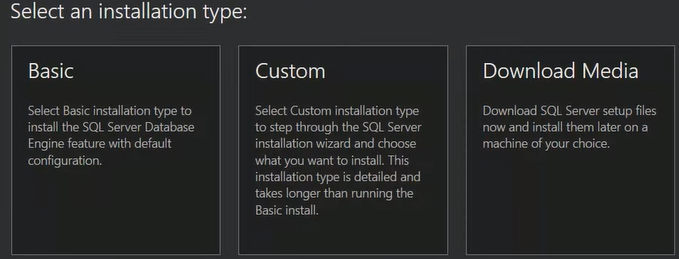
Step 1: Open chrome & paste below link

<https://www.microsoft.com/en-in/sql-server/sql-server-downloads>

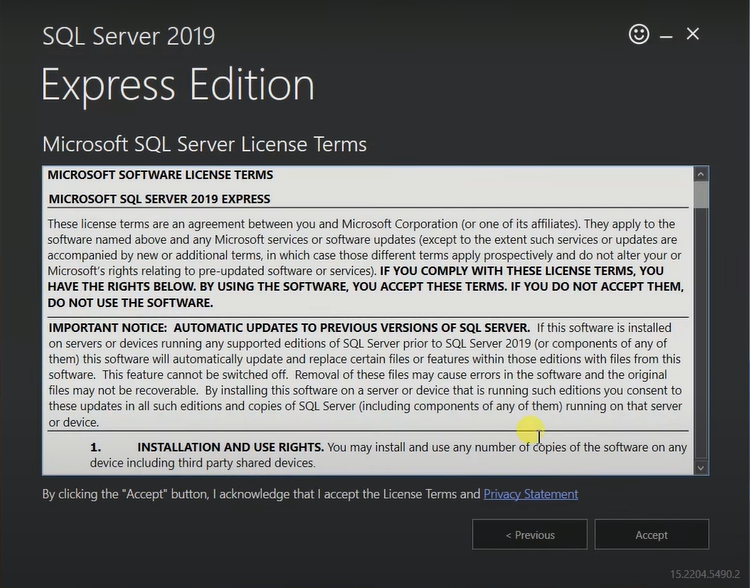
Step 2: Select Express one below snippet given



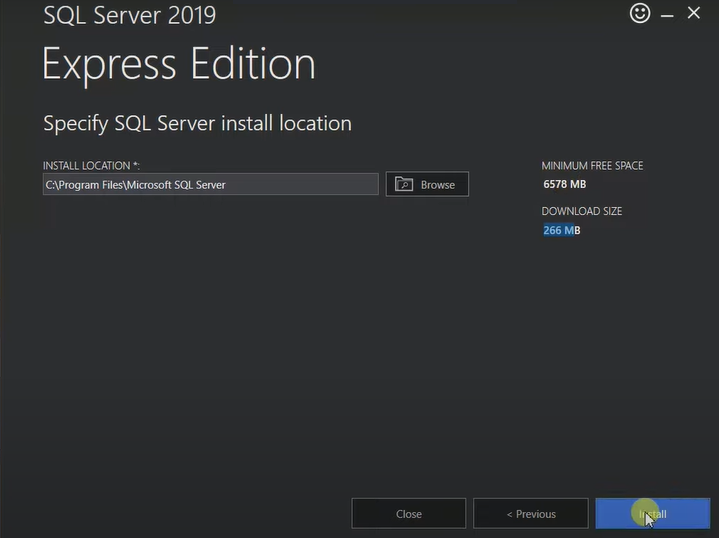
Step 3: Once Express is downloaded & while installing select Basic below is snippet



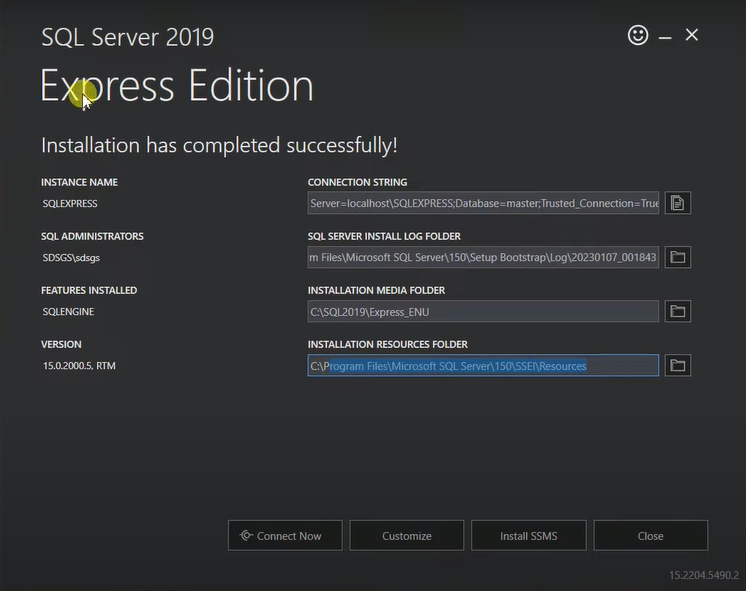
Step 4: Click in accept snippet given below:



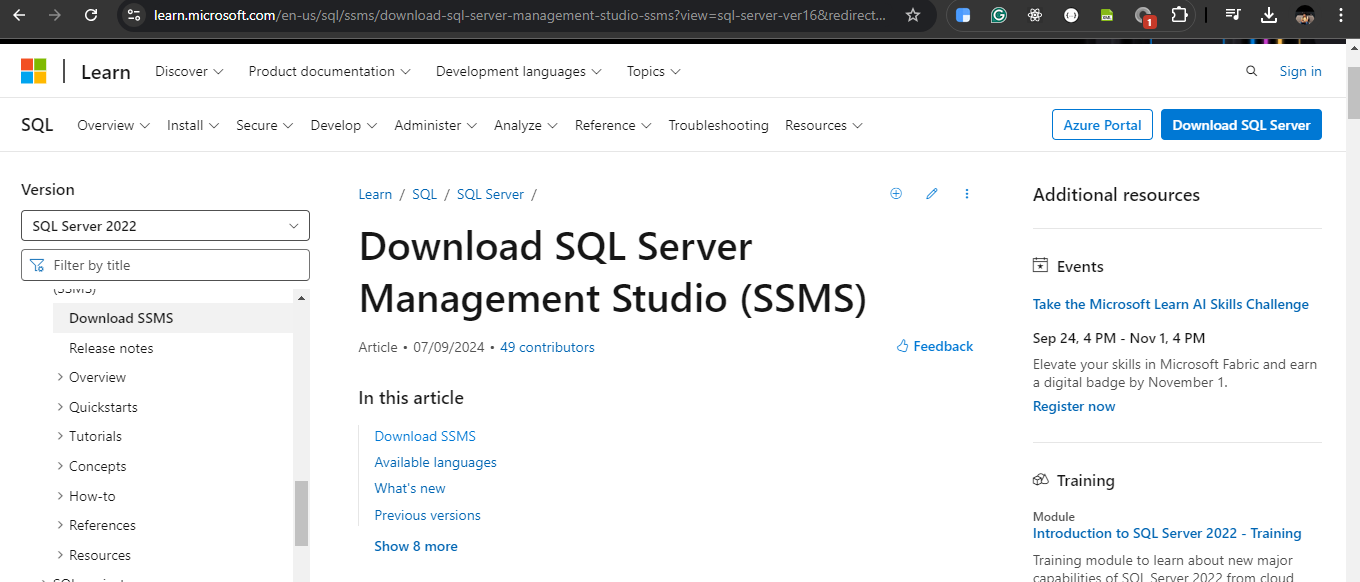
Step 5: Click on install & wait for few minutes



Step 6: Congratulation you SQL Server installed successfully below snippet you can see now click on install SSMS (SQL Server Management Studio) when you will click on install SSMS it will redirect to below link:



<https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16&redirectedfrom=MSDN>

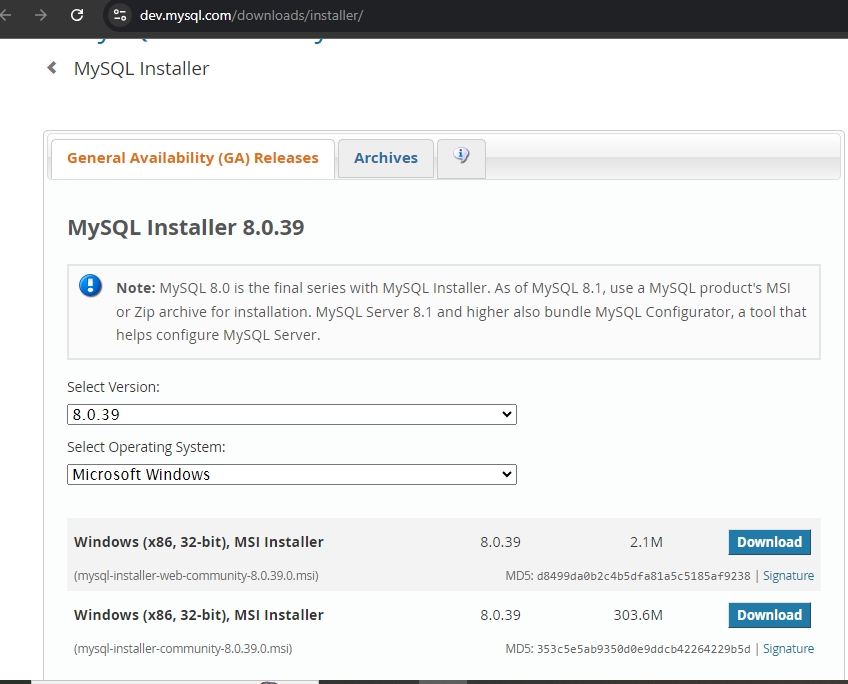


Step 7: Download SSMS and install it now you can you SQL Server Management Studio to write your SQL Query to interact with database

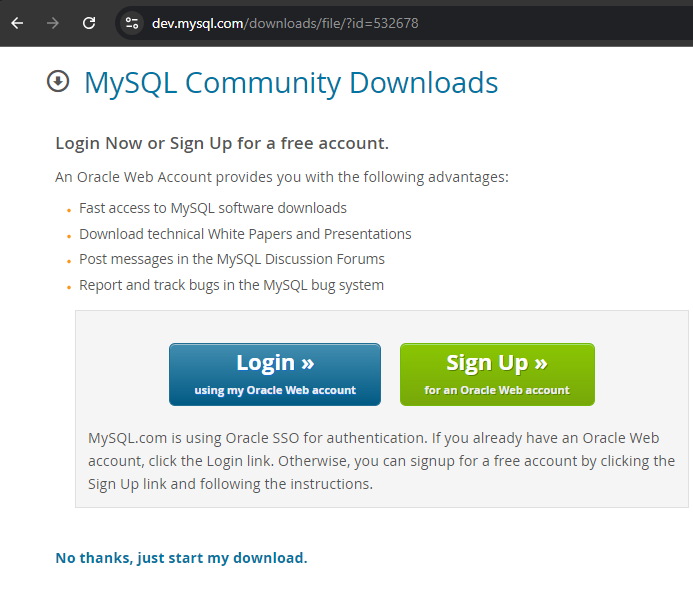
Download and Install MySql Database guide:-

Step 1: Open this link <https://dev.mysql.com/downloads/installer/>

Step 2: Select click on MySQL community Installer MSI below snippet given



Step 3: Click on No Thanks just start my download snippet given below:



Next steps refer from here <https://www.javatpoint.com/how-to-install-mysql>

**Difference Between ALTER & UPDATE**

**1. UPDATE**

* **Purpose**: Modifies the data within an existing table.
* **Used For**: Updating rows in a table.
* **Syntax**:

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

* **Example**:

UPDATE employees

SET salary = 60000

WHERE employee\_id = 5;

* + This updates the salary of the employee with employee\_id = 5.

**2. ALTER**

* **Purpose**: Modifies the structure of an existing table (e.g., adding, deleting, or modifying columns).
* **Used For**: Altering the schema or structure of a table.
* **Syntax**:
  + Add a new column:

ALTER TABLE table\_name

ADD column\_name datatype;

* + Modify an existing column:

ALTER TABLE table\_name

MODIFY column\_name datatype;

* + Drop a column:

ALTER TABLE table\_name

DROP COLUMN column\_name;

* **Example**:

ALTER TABLE employees

ADD phone\_number VARCHAR(15);

* + This adds a new column phone\_number to the employees table.

**Summary:**

* UPDATE changes the **data** inside the table.
* ALTER changes the **structure** of the table (like columns or constraints).

Top of Form

**Difference Between ALTER & RENAME**

**ALTER**

* **Purpose**: Modifies the **structure** of an existing table (e.g., adding, deleting, or modifying columns, changing constraints).
* **Used For**: Changing the table schema (columns, data types, constraints, etc.).
* **Syntax**:
  + Add a column:

ALTER TABLE table\_name

ADD column\_name datatype;

* + Modify a column:

ALTER TABLE table\_name

MODIFY column\_name datatype;

* + Drop a column:

ALTER TABLE table\_name

DROP COLUMN column\_name;

* + Add or drop constraints:

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name ...;

ALTER TABLE table\_name

DROP CONSTRAINT constraint\_name;

* **Example**:

ALTER TABLE employees

ADD phone\_number VARCHAR(15);

* + This adds a new column phone\_number to the employees table.

**2. RENAME**

* **Purpose**: Renames **database objects** such as tables or columns.
* **Used For**: Changing the name of tables, columns, or constraints.
* **Syntax**:
  + Rename a table:

RENAME TABLE old\_table\_name TO new\_table\_name;

* + Rename a column (syntax varies across databases):
    - In MySQL:

ALTER TABLE table\_name

RENAME COLUMN old\_column\_name TO new\_column\_name;

* + - In Oracle:

ALTER TABLE table\_name

RENAME COLUMN old\_column\_name TO new\_column\_name;

* **Example**:

RENAME TABLE employees TO staff;

* + This renames the employees table to staff.

**Key Differences:**

* **ALTER** changes the structure of a table (add/drop/modify columns, constraints).
* **RENAME** changes the name of the table or its columns, but does not affect the structure or data in the table.

**Summary:**

* Use ALTER to **modify the table structure**.
* Use RENAME to **change the name** of the table or columns.

Refer this resources for better understanding <https://prepinsta.com/dbms/alter/>

Bottom of Form

**Data Types in SQL:**

**1. INT (Integer)**

* **Definition**: Stores whole numbers (without decimals).
* **Example**:

employee\_id INT;

* **When to Use**: Use INT for storing numeric values that do not require decimals, such as IDs, age, quantity, and counts.

**2. DECIMAL (or NUMERIC)**

* **Definition**: Stores fixed-point numbers, useful for precise arithmetic calculations. It has two parts: precision (total number of digits) and scale (number of digits after the decimal point).
* **Example**:

salary DECIMAL(10, 2);

* **When to Use**: Use DECIMAL for financial data like prices, salaries, or any numbers that need exact precision. E.g., monetary values.

**3. VARCHAR (Variable-Length Character String)**

* **Definition**: Stores variable-length strings, where the length can vary up to a specified limit.
* **Example**:

email VARCHAR(255);

* **When to Use**: Use VARCHAR for text data where the length can vary, like names, addresses, or emails. Ideal for storing textual data that doesn't have a fixed length.

**4. CHAR (Fixed-Length Character String)**

* **Definition**: Stores fixed-length strings. If the data is shorter than the defined length, it is padded with spaces.
* **Example**:

status CHAR(1);

* **When to Use**: Use CHAR for fixed-length text fields, such as single-character status codes (like 'M' for Male or 'F' for Female), country codes, names, addresses, or emails. Ideal for storing textual data etc.

**5. DATE**

* **Definition**: Stores a date (without time), typically in the format YYYY-MM-DD.
* **Example**:

birthdate DATE;

* **When to Use**: Use DATE for storing only date information, such as birthdays, order dates, or registration dates.

**6. DATETIME**

* **Definition**: Stores both date and time, typically in the format YYYY-MM-DD HH:MM:SS.
* **Example**:

created\_at DATETIME;

* **When to Use**: Use DATETIME when you need to store both date and time information, like timestamps for when a record is created or modified.

**7. TIMESTAMP**

* **Definition**: Stores date and time. Automatically updates to the current time when the record is modified, depending on the SQL engine.
* **Example**:

last\_modified TIMESTAMP;

* **When to Use**: Use TIMESTAMP to store data that requires automatic tracking of changes, like last updated records or creation timestamps.

**8. FLOAT/REAL**

* **Definition**: Stores approximate floating-point numbers. Useful for scientific calculations but may lose precision.
* **Example**:

rating FLOAT;

* **When to Use**: Use FLOAT when you need to store large or small real numbers, such as scientific measurements or large decimal values that do not need absolute precision.

**9. BOOLEAN (or BIT)**

* **Definition**: Stores true/false values or 0/1 (depending on the database system).
* **Example**:

is\_active BOOLEAN;

* **When to Use**: Use BOOLEAN for true/false values, such as flags for user activity or subscription status.

**10. TEXT**

* **Definition**: Stores large amounts of variable-length text.
* **Example**:

description TEXT;

* **When to Use**: Use TEXT for storing long, unstructured textual data like product descriptions, user comments, or logs.

**11. BLOB (Binary Large Object)**

* **Definition**: Stores large binary data like images, audio, video, or other multimedia files.
* **Example**:

image BLOB;

* **When to Use**: Use BLOB for binary data like images, files, or any other large non-textual data.

**12. ENUM**

* **Definition**: Allows a predefined set of values for a column.
* **Example**:

gender ENUM('Male', 'Female', 'Other');

* **When to Use**: Use ENUM when a column must contain one of a set of predefined values, like gender, status, or categories.

**13. BIGINT**

* **Definition**: Stores large integer values, useful when INT is not large enough.
* **Example**:

transaction\_id BIGINT;

* **When to Use**: Use BIGINT when storing very large numbers, like transaction IDs or high-precision counters.

**14. SMALLINT**

* **Definition**: Stores small integer values.
* **Example**:

age SMALLINT;

* **When to Use**: Use SMALLINT when storing smaller whole numbers to save space, like small counts or ages.

**15. NVARCHAR**

**When to Use NVARCHAR**

* **Internationalization**: When your application needs to support multiple languages, use NVARCHAR to ensure proper storage of characters.
* **Special Characters**: If you expect to store text that includes special characters (like emojis, accented characters, or symbols), NVARCHAR is a good choice.

#### **Creating a Table with NVARCHAR Columns**

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

first\_name NVARCHAR(50),

last\_name NVARCHAR(50),

email NVARCHAR(100),

phone\_number NVARCHAR(20),

address NVARCHAR(255)

);

#### **Inserting Data into the NVARCHAR Columns**

-- Inserting data into the employees table

INSERT INTO employees (employee\_id, first\_name, last\_name, email, phone\_number, address)

VALUES

(1, N'Алексей', N'Иванов', N'aleksey@example.com', N'+7 123 456 7890', N'Москва, Россия'),

(2, N'John', N'Doe', N'john.doe@example.com', N'123-456-7890', N'123 Elm St, Springfield, IL'),

(3, N'张伟', N'李', N'zhang.wei@example.com', N'+86 123 456 7890', N'北京市, 中国');

**Summary of When to Use Each Data Type:**

* **INT, BIGINT, SMALLINT**: When storing numeric values without decimal points.
* **DECIMAL**: When storing precise decimal values (like money or financial data).
* **VARCHAR, CHAR**: When storing text (use VARCHAR for variable-length, CHAR for fixed-length).
* **DATE, DATETIME, TIMESTAMP**: When storing date or time information.
* **FLOAT**: For approximate values with a wide range but lower precision.
* **BOOLEAN**: For true/false or binary values.
* **TEXT, BLOB**: For large text or binary data.
* **ENUM**: For predefined choices in a column.

**For more information can refer this resource** [**https://www.w3schools.com/sql/sql\_datatypes.asp**](https://www.w3schools.com/sql/sql_datatypes.asp)

**Operators in SQL:**

### SQL Arithmetic Operators

SQL provides several arithmetic operators that can be used to perform mathematical operations on numeric data types. These operators allow you to manipulate data effectively within your SQL queries.

#### 1. Addition (+)

* **Description:** Adds two numeric values.
* **Example:**

SELECT 10 + 5 AS Sum; -- Result: 15

#### 2. Subtraction (-)

* **Description:** Subtracts one numeric value from another.
* **Example:**

SELECT 10 - 5 AS Difference; -- Result: 5

#### 3. Multiplication (\*)

* **Description:** Multiplies two numeric values.
* **Example:**

SELECT 10 \* 5 AS Product; -- Result: 50

#### 4. Division (/)

* **Description:** Divides one numeric value by another. Note that dividing by zero will result in an error.
* **Example:**

SELECT 10 / 5 AS Quotient; -- Result: 2

#### 5. Modulus (%)

* **Description:** Returns the remainder of a division operation.
* **Example:**

SELECT 10 % 3 AS Remainder; -- Result: 1

### SQL Comparison Operators

SQL comparison operators are used to compare two values. They return a boolean result (TRUE, FALSE, or UNKNOWN) based on the comparison. These operators are essential for filtering records in SQL queries.

#### Common Comparison Operators

1. **Equal to (=)**
   * **Description:** Checks if two values are equal.
   * **Example:**

SELECT \* FROM Employees WHERE Salary = 50000;

1. **Not equal to (!= or <>)**
   * **Description:** Checks if two values are not equal.
   * **Example:**

SELECT \* FROM Employees WHERE Salary <> 50000; -- Using <>

-- OR

SELECT \* FROM Employees WHERE Salary != 50000; -- Using !=

1. **Greater than (>)**
   * **Description:** Checks if the left value is greater than the right value.
   * **Example:**

SELECT \* FROM Employees WHERE Salary > 50000;

1. **Less than (<)**
   * **Description:** Checks if the left value is less than the right value.
   * **Example:**

SELECT \* FROM Employees WHERE Salary < 50000;

1. **Greater than or equal to (>=)**
   * **Description:** Checks if the left value is greater than or equal to the right value.
   * **Example:**

SELECT \* FROM Employees WHERE Salary >= 50000;

1. **Less than or equal to (<=)**
   * **Description:** Checks if the left value is less than or equal to the right value.
   * **Example:**

SELECT \* FROM Employees WHERE Salary <= 50000;

1. **BETWEEN**
   * **Description:** Checks if a value falls within a specified range.
   * **Example:**

SELECT \* FROM Employees WHERE Salary BETWEEN 40000 AND 60000;

1. **LIKE**
   * **Description:** Used to search for a specified pattern in a column (often with wildcard characters).
   * **Example:**

SELECT \* FROM Employees WHERE FirstName LIKE 'J%'; -- Names starting with 'J'

1. **IN**
   * **Description:** Checks if a value matches any value in a specified list.
   * **Example:**

SELECT \* FROM Employees WHERE EmployeeID IN (1, 2, 3);

1. **IS NULL**
   * **Description:** Checks if a value is NULL.
   * **Example:**

SELECT \* FROM Employees WHERE Bonus IS NULL;

1. **IS NOT NULL**
   * **Description:** Checks if a value is NOT NULL.
   * **Example:**

SELECT \* FROM Employees WHERE Bonus IS NOT NULL;

### Using Comparison Operators in Queries

Comparison operators are widely used in SQL queries to filter and retrieve specific records based on conditions. Below are some examples demonstrating their usage:

#### Example Table: Employees

| **EmployeeID** | **FirstName** | **LastName** | **Salary** |
| --- | --- | --- | --- |
| 1 | John | Doe | 50000 |
| 2 | Jane | Smith | 60000 |
| 3 | Alice | Johnson | 55000 |
| 4 | Bob | Brown | 45000 |
| 5 | Charlie | Davis | NULL |

#### Examples of Queries Using Comparison Operators

1. **Select Employees with Salary Greater than 50000**

SELECT \* FROM Employees WHERE Salary > 50000;

1. **Select Employees with Salary Less than or Equal to 50000**

SELECT \* FROM Employees WHERE Salary <= 50000;

1. **Select Employees whose First Names Start with 'A'**

SELECT \* FROM Employees WHERE FirstName LIKE 'A%';

1. **Select Employees with Salary in a Specific Range**

SELECT \* FROM Employees WHERE Salary BETWEEN 45000 AND 60000;

1. **Select Employees with a Specific EmployeeID**

SELECT \* FROM Employees WHERE EmployeeID IN (1, 3);

1. **Select Employees with NULL Bonus Values**

SELECT \* FROM Employees WHERE Bonus IS NULL;

### SQL Logical Operators

SQL logical operators are used to combine multiple conditions in a query. They allow you to construct complex queries by evaluating multiple expressions. The primary logical operators in SQL are AND, OR, and NOT.

#### 1. AND

* **Description:** The AND operator combines two or more conditions and returns TRUE only if all conditions are true.
* **Syntax:**

SELECT column1, column2, ...

FROM table\_name

WHERE condition1 AND condition2;

* **Example:**

SELECT \* FROM Employees

WHERE Salary > 50000 AND Department = 'Sales';

This query retrieves employees with a salary greater than 50,000 who work in the Sales department.

#### 2. OR

* **Description:** The OR operator combines two or more conditions and returns TRUE if at least one condition is true.
* **Syntax:**

SELECT column1, column2, ...

FROM table\_name

WHERE condition1 OR condition2;

* **Example:**

SELECT \* FROM Employees

WHERE Salary < 40000 OR Department = 'HR';

This query retrieves employees who have a salary less than 40,000 or work in the HR department.

#### 3. NOT

* **Description:** The NOT operator negates a condition, returning TRUE if the condition is false.
* **Syntax:**

SELECT column1, column2, ...

FROM table\_name

WHERE NOT condition;

* **Example:**

SELECT \* FROM Employees

WHERE NOT Department = 'Finance';

This query retrieves employees who do not work in the Finance department.

### Combining Logical Operators

You can combine logical operators to create more complex conditions using parentheses to group conditions as needed.

#### Example of Combined Conditions

SELECT \* FROM Employees

WHERE (Salary > 50000 AND Department = 'Sales') OR (Department = 'HR' AND NOT Bonus IS NULL);

This query retrieves employees who either have a salary greater than 50,000 and work in Sales, or work in HR and have a non-null Bonus.

### Practical Examples

#### Example Table: Employees

| **EmployeeID** | **FirstName** | **LastName** | **Salary** | **Department** | **Bonus** |
| --- | --- | --- | --- | --- | --- |
| 1 | John | Doe | 50000 | Sales | 5000 |
| 2 | Jane | Smith | 60000 | HR | NULL |
| 3 | Alice | Johnson | 55000 | IT | 3000 |
| 4 | Bob | Brown | 45000 | Finance | 2000 |
| 5 | Charlie | Davis | 40000 | Sales | NULL |

#### Examples of Queries Using Logical Operators

1. **Select Employees in Sales or IT with Salary Greater than 50000**

SELECT \* FROM Employees

WHERE (Department = 'Sales' OR Department = 'IT') AND Salary > 50000;

1. **Select Employees who have a Bonus or are in HR**

SELECT \* FROM Employees

WHERE Bonus IS NOT NULL OR Department = 'HR';

1. **Select Employees whose Salary is less than 50000 and do not work in Finance**

SELECT \* FROM Employees

WHERE Salary < 50000 AND NOT Department = 'Finance';

**Types of SQL Constraints**

1. **NOT NULL**
2. **UNIQUE**
3. **PRIMARY KEY**
4. **FOREIGN KEY**
5. **CHECK**
6. **DEFAULT**
7. **INDEX** (not a constraint in all contexts but often mentioned)

**1. NOT NULL**

* **Definition:** Ensures that a column cannot have a NULL value.
* **Usage:** When you want to make sure that every row in the table has a value for that column.
* **Example:**

CREATE TABLE Employees (

EmployeeID INT NOT NULL,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50),

Age INT

);

*In this example, EmployeeID and FirstName must have values for each employee.*

**2. UNIQUE**

* **Definition:** Ensures that all values in a column are different.
* **Usage:** When you need to maintain unique values for a column but do not want to make it a primary key.
* **Example:**

CREATE TABLE Employees (

EmployeeID INT NOT NULL,

Email VARCHAR(100) UNIQUE,

FirstName VARCHAR(50),

LastName VARCHAR(50)

);

*In this example, no two employees can have the same email address.*

**3. PRIMARY KEY**

* **Definition:** A combination of NOT NULL and UNIQUE. It uniquely identifies each record in a table.
* **Usage:** When you need a unique identifier for each row in the table.
* **Example:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50)

);

*Here, EmployeeID serves as the unique identifier for each employee.*

**4. FOREIGN KEY**

* **Definition:** A key used to link two tables together. It ensures referential integrity between the two tables.
* **Usage:** When you want to enforce a relationship between two tables.
* **Example:**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

*In this example, DepartmentID in the Employees table references the DepartmentID in the Departments table, ensuring that each employee belongs to a valid department.*

**5. CHECK**

* **Definition:** Ensures that all values in a column satisfy a specific condition.
* **Usage:** When you want to enforce a certain range of values or a specific condition for a column.
* **Example:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Age INT CHECK (Age >= 18)

);

*This example ensures that all employees must be at least 18 years old.*

**6. DEFAULT**

* **Definition:** Provides a default value for a column when none is specified during the insertion of a new record.
* **Usage:** When you want to set a default value for a column.
* **Example:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

HireDate DATE DEFAULT CURRENT\_DATE

);

*Here, if no HireDate is specified, it will default to the current date.*

**7. INDEX**

* **Definition:** An index improves the speed of data retrieval operations on a database table at the cost of additional space and decreased performance on data modification.
* **Usage:** When you want to improve query performance.
* **Example:**

CREATE INDEX idx\_lastname ON Employees (LastName);

*This creates an index on the LastName column to speed up queries that filter by last name.*

**Summary**

SQL constraints are essential for ensuring data integrity and enforcing rules at the database level. Here’s a quick overview of their functionalities:

* **NOT NULL:** Prevents NULL values.
* **UNIQUE:** Ensures all values in a column are unique.
* **PRIMARY KEY:** Combines NOT NULL and UNIQUE; uniquely identifies a row.
* **FOREIGN KEY:** Ensures referential integrity between two tables.
* **CHECK:** Validates data against a condition.
* **DEFAULT:** Sets a default value for a column.
* **INDEX:** Enhances query performance.

**SQL Functions:**

SQL functions are essential for performing operations on data stored in a database. They can be categorized into two main types: **Scalar Functions** and **Aggregate Functions**. Below is a comprehensive overview of SQL functions, including their definitions, examples, and usage.

**Types of SQL Functions**

1. **Scalar Functions**
2. **Aggregate Functions**
3. **String Functions**
4. **Numeric Functions**
5. **Date Functions**
6. **Conversion Functions**
7. **User-Defined Functions (UDF)**

**1. Scalar Functions**

**Definition:** Scalar functions return a single value for each row processed and operate on a single value. Examples include string manipulation, mathematical calculations, and date/time operations.

**Examples:**

* **UPPER()**: Converts a string to uppercase.

SELECT UPPER(FirstName) AS UpperFirstName FROM Employees;

* **LOWER()**: Converts a string to lowercase.

SELECT LOWER(LastName) AS LowerLastName FROM Employees;

* **ROUND()**: Rounds a numeric value to a specified number of decimal places.

SELECT ROUND(Salary, 2) AS RoundedSalary FROM Employees;

* **LENGTH()**: Returns the length of a string.

SELECT LENGTH(FirstName) AS FirstNameLength FROM Employees;

**2. Aggregate Functions**

**Definition:** Aggregate functions perform calculations on a set of values and return a single value. These are often used with the GROUP BY clause.

**Examples:**

* **COUNT()**: Returns the number of rows that match a specified criterion.

SELECT COUNT(\*) AS TotalEmployees FROM Employees;

* **SUM()**: Returns the total sum of a numeric column.

SELECT SUM(Salary) AS TotalSalary FROM Employees;

* **AVG()**: Returns the average value of a numeric column.

SELECT AVG(Salary) AS AverageSalary FROM Employees;

* **MIN()**: Returns the minimum value in a set.

SELECT MIN(Age) AS YoungestEmployee FROM Employees;

* **MAX()**: Returns the maximum value in a set.

SELECT MAX(Salary) AS HighestSalary FROM Employees;

**3. String Functions**

String functions are a subset of scalar functions specifically designed for manipulating string data.

**Examples:**

* **CONCAT()**: Concatenates two or more strings.

SELECT CONCAT(FirstName, ' ', LastName) AS FullName FROM Employees;

* **SUBSTRING()**: Extracts a substring from a string.

SELECT SUBSTRING(Email, 1, 5) AS EmailPrefix FROM Employees;

* **TRIM()**: Removes leading and trailing spaces from a string.

SELECT TRIM(FirstName) AS TrimmedFirstName FROM Employees;

**4. Numeric Functions**

Numeric functions perform mathematical calculations and operations.

**Examples:**

* **ABS()**: Returns the absolute value of a number.

SELECT ABS(-25) AS AbsoluteValue;

* **CEIL()**: Returns the smallest integer value that is greater than or equal to a number.

SELECT CEIL(4.3) AS CeilingValue; -- Returns 5

* **FLOOR()**: Returns the largest integer value that is less than or equal to a number.

SELECT FLOOR(4.8) AS FloorValue; -- Returns 4

**5. Date Functions**

Date functions are used to manipulate and extract information from date and time values.

**Examples:**

* **NOW()**: Returns the current date and time.

SELECT NOW() AS CurrentDateTime;

* **DATEADD()**: Adds a specified interval to a date.

SELECT DATEADD(YEAR, 1, HireDate) AS NextYearHireDate FROM Employees;

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

SELECT DATEDIFF(NOW(), HireDate) AS DaysSinceHired FROM Employees;

**6. Conversion Functions**

Conversion functions are used to convert data from one type to another.

**Examples:**

* **CAST()**: Converts a value to a specified data type.

SELECT CAST(Salary AS VARCHAR(10)) AS SalaryAsString FROM Employees;

* **CONVERT()**: Converts a value from one type to another (specific to some SQL databases).

SELECT CONVERT(DATE, HireDate) AS ConvertedHireDate FROM Employees;

**7. User-Defined Functions (UDF)**

**Definition:** User-defined functions allow you to create your own functions to perform specific calculations or operations that can be reused.

**Example:**

* Creating a simple user-defined function that returns the full name of an employee.

CREATE FUNCTION GetFullName(@FirstName VARCHAR(50), @LastName VARCHAR(50))

RETURNS VARCHAR(100)

AS

BEGIN

RETURN CONCAT(@FirstName, ' ', @LastName);

END;

* Using the function:

SELECT dbo.GetFullName(FirstName, LastName) AS FullName FROM Employees;

**Summary**

SQL functions are powerful tools that allow you to manipulate and process data efficiently. They can be broadly classified into scalar functions, aggregate functions, string functions, numeric functions, date functions, conversion functions, and user-defined functions.