## 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.

## 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

## 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.