**Structured Query Language (SQL)**

 SQL is a standardized language for managing and manipulating relational databases. It enables efficient querying, updating, and managing of database structures.

**Why Learn SQL?**

* **Universal Usage:** Compatible with major database systems like MySQL, PostgreSQL, Oracle, and SQL Server.
* **High Demand:** Essential for careers in data analysis, software development, and database administration.
* **Versatility:** Applicable across various industries for effective data management and analysis.
* **Efficiency:** Facilitates quick and reliable data retrieval and manipulation.

**SQL Data Types**

Data types in SQL define the kind of data that can be stored in a column.

**Common SQL Data Types:**

1. **INT**: Stores whole numbers.
   * Example: Age INT
2. **VARCHAR(n)**: Stores variable-length strings.
   * Example: Name VARCHAR(50)
3. **CHAR(n)**: Stores fixed-length strings.
   * Example: Code CHAR(3)
4. **DECIMAL(p, s)**: Stores precise numeric values.
   * Example: Price DECIMAL(10,2)
   * p: Total digits, s: Digits after the decimal point.
5. **FLOAT**: Stores approximate numeric values.
   * Example: Rate FLOAT
6. **DATE**: Stores dates in YYYY-MM-DD format.
   * Example: BirthDate DATE
7. **DATETIME**: Stores date and time.
   * Example: OrderTime DATETIME
8. **TEXT**: Stores large blocks of text.
   * Example: Description TEXT
9. **BOOLEAN**: Stores TRUE or FALSE.
   * Example: IsActive BOOLEAN

COMMANDS IN SQL

**DDL Commands**

**Data Definition Language (DDL)** commands define and modify the structure of database objects, directly impacting the database schema.

**1. CREATE**

The CREATE command is used to establish new database objects like tables, indexes, or views.

* **Creating a Table:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50) NOT NULL,

HireDate DATE,

Salary DECIMAL(10, 2)

);

* + **EmployeeID INT:** Unique identifier for each employee.
  + **Name VARCHAR(50):** Employee's name, up to 50 characters.
  + **HireDate DATE:** Hiring date in YYYY-MM-DD format.
  + **Salary DECIMAL(10, 2):** Salary with up to 10 digits and 2 decimal places.
* **Creating an Index:**

CREATE INDEX idx\_name ON Employees(Name);

* + **idx\_name:** Name of the index.
  + **Employees(Name):** Index on the Name column of the Employees table.

**2.** **ALTER**

The ALTER command modifies the structure of an existing database object, such as adding or deleting columns.

* **Adding a Column:**

 ALTER TABLE Employees ADD Department VARCHAR(50);

* **Modifying a Column:**

ALTER TABLE Employees MODIFY Salary DECIMAL(12, 2);

* **Dropping a Column:**

ALTER TABLE Employees DROP COLUMN Department;

**3. DROP**

 The DROP command permanently deletes database objects. Use it cautiously as it removes the object and its data.

**Dropping a Table:**

 DROP TABLE Employees;

**Dropping an Index:**

 DROP INDEX idx\_name ON Employees;

**4. TRUNCATE**

 The TRUNCATE command removes all records from a table but retains its structure for future use. It is faster than DELETE as it doesn’t log individual row deletions.

**Truncating a Table:**

 TRUNCATE TABLE Employees;

**Key Points:**

* Does not affect the table schema or its constraints.
* Resets any auto-increment counters.
* DROP removes the table and its structure permanently, while TRUNCATE deletes all rows but retains the table structure for future use.

**Data Manipulation Language (DML)**

DML is a subset of SQL used for managing and manipulating data within existing database tables.

* **Purpose:** Allows users to insert, update, delete, and retrieve data in database tables.

**Common DML Operations:**

1. **INSERT** – Adds new data into a table.
2. **UPDATE** – Modifies existing data.
3. **DELETE** – Removes data from a table.
4. **SELECT** – Retrieves data from one or more tables.

**1. INSERT Command**

  The **INSERT** statement is used to add new rows of data into a table. You can insert a single record or multiple records at once.

**Key Characteristics:**

* You must specify the values to be inserted for each column or specify only the columns to be inserted (leaving others to default).
* The number of values inserted must match the number of columns specified.

**Syntax:**

 INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

**Example:**

 INSERT INTO Employees (EmployeeID, Name, HireDate, Salary)

VALUES (101, 'John Doe', '2023-01-01', 50000.00);

Here, EmployeeID, Name, HireDate, and Salary are the columns, and (101, 'John Doe', '2023-01-01', 50000.00) are the values inserted into those columns.

**Multiple Insert Example:**

 INSERT INTO Employees (EmployeeID, Name, HireDate, Salary)

VALUES (102, 'Jane Smith', '2023-02-01', 60000.00),

(103, 'Alice Brown', '2023-03-01', 55000.00);

**2. UPDATE Command**

 The **UPDATE** statement is used to modify existing records in a table. It can update one or more columns of one or more records.

**Key Characteristics:**

* Always use the WHERE clause to avoid updating all records in the table.
* If no WHERE clause is provided, all rows will be updated.

**Syntax:**

 UPDATE table\_name

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

WHERE condition;

**Example:**

 UPDATE Employees

SET Salary = 55000

WHERE EmployeeID = 101;

In this example, the Salary for the employee with EmployeeID 101 is updated to 55000.

**3. DELETE Command**

 The **DELETE** statement is used to remove one or more rows from a table. It does not remove the table structure, only the data.

**Key Characteristics:**

* Similar to UPDATE, always use the WHERE clause to delete specific rows.
* If no WHERE clause is included, **all records** in the table will be deleted.

**Syntax:**

DELETE FROM table\_name

WHERE condition;

**Example:**

 DELETE FROM Employees

WHERE EmployeeID = 102;

This command deletes the employee with EmployeeID 102 from the Employees table.

**Delete All Records:**

 DELETE FROM Employees;

 This deletes all rows but keeps the table structure intact.

**Data Control Language (DCL)**

 DCL consists of SQL commands used to manage permissions and access rights within a database, ensuring authorized access and maintaining integrity.

**Key Components of DCL**

**1. GRANT**

**Definition:** The GRANT command assigns specific privileges to users or roles, allowing them to perform certain actions on database objects.

**Usage:**

**Granting Permissions:** Allows users to execute specific commands like SELECT, INSERT, UPDATE, or DELETE on tables, views, or other objects.

**Example:**

  GRANT SELECT, INSERT ON Employees TO UserA;

**2. REVOKE**

**Definition:** The REVOKE command removes previously granted privileges from users or roles, restricting their access to database objects.

**Usage:**

**Revoking Permissions:** Removes specific permissions, ensuring that users no longer have access to perform certain actions.

**Example:**

REVOKE INSERT ON Employees FROM UserA;

**Transaction Control Language (TCL)**

* **Manages Transactions:** Handles sequences of operations as single units to ensure data consistency.
* **Ensures Data Integrity:** Maintains reliable and consistent data, especially in multi-user environments.

**Key Components of TCL**

* **COMMIT**
* **ROLLBACK**
* **SAVEPOINT**

**COMMIT**

**Definition:** The COMMIT command saves all changes made during the current transaction permanently to the database.

**Usage:**

**Finalizing Transactions:** Ensures that all operations within the transaction are completed successfully and the changes are stored.

**Example:**

 BEGIN TRANSACTION;

UPDATE Accounts

SET Balance = Balance - 500

WHERE AccountID = 1;

UPDATE Accounts

SET Balance = Balance + 500

WHERE AccountID = 2;

COMMIT;

**ROLLBACK**

**Definition:** The ROLLBACK command undoes all changes made during the current transaction, reverting the database to its previous state.

**Usage:**

**Error Handling:** Reverts changes if an error occurs during the transaction, maintaining data integrity.

**Example:**

 BEGIN TRANSACTION;

UPDATE Accounts

SET Balance = Balance - 500

WHERE AccountID = 1;

-- An error occurs here

ROLLBACK;

**SAVEPOINT**

**Definition:** The SAVEPOINT command sets a point within a transaction to which you can later roll back without affecting the entire transaction.

**Usage:**

**Partial Rollbacks:** Allows reverting specific parts of a transaction while keeping other changes intact.

**Example:**

 BEGIN TRANSACTION;

UPDATE Accounts

SET Balance = Balance - 500

WHERE AccountID = 1;

SAVEPOINT DebitCompleted;

UPDATE Accounts

SET Balance = Balance + 500

WHERE AccountID = 2;

-- If an error occurs after the first update

ROLLBACK TO SAVEPOINT DebitCompleted;

COMMIT;

**SQL Constraints**

 SQL constraints are rules applied to table columns to maintain data accuracy and consistency, enforcing business logic and preventing incorrect data modifications.

* **Common Constraints:** Include **PRIMARY KEY**, **FOREIGN KEY**, **UNIQUE**, **CHECK**, and **DEFAULT**, each serving specific roles in maintaining the database's integrity and structure.

**1. PRIMARY KEY**

* **Definition:** A **PRIMARY KEY** uniquely identifies each record in a table. It must contain unique, non-null values and ensures that each entry can be distinctly retrieved.
* **Example:**

 CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50),

HireDate DATE);

**2. FOREIGN KEY**

A **FOREIGN KEY** links two tables by matching values in the foreign key column with those in the referenced table, ensuring referential integrity.

* **Cascading Actions:** Supports operations like ON DELETE CASCADE or ON UPDATE CASCADE to automatically update or delete related rows.

**Example:**

 CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

EmployeeID INT,

OrderDate DATE,

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID));

In this example, EmployeeID in the Orders table is a foreign key referencing EmployeeID in the Employees table.

**3. UNIQUE**

 The **UNIQUE** constraint ensures all values in a column are distinct, allowing no duplicate entries.

* Unlike a primary key, it permits **NULL** values, but each non-NULL value must still be unique.

**Key Characteristics:**

* **Uniqueness:** Ensures that no two rows can have the same value in the unique column.
* **Multiple Unique Constraints:** A table can have multiple unique constraints.

**Example:**

 CREATE TABLE Users (

UserID INT PRIMARY KEY,

Email VARCHAR(100) UNIQUE);

 Here, Email must be unique for each user, ensuring no two users can have the same email address.

**4. CHECK**

The **CHECK constraint** enforces a condition that values in a column must meet, ensuring only valid data is inserted.

* **Purpose:** Maintains domain integrity by restricting data to a specified range or format.

**Key Characteristics:**

* **Validation:** Ensures that values meet a specific condition before being inserted or updated.
* **Custom Conditions:** You can use operators like >, <, =, or BETWEEN in the condition.

**Example:**

 CREATE TABLE Employees ( EmployeeID INT PRIMARY KEY, Age INT CHECK (Age >= 18),

Salary DECIMAL(10, 2));

 Here, the CHECK constraint ensures that the Age is at least 18, enforcing that only employees aged 18 or older can be added.

**5. DEFAULT**

 The **DEFAULT** constraint provides a default value for a column when no value is specified during an INSERT operation.

**Key Characteristics:**

* **Automatic Assignment:** Automatically assigns a value if no value is provided.
* **Applicable to NULL Values:** If a column is defined with a DEFAULT constraint, it will use the default value when no explicit value is supplied during insertion.

**Example:**

 CREATE TABLE Products (

ProductID INT PRIMARY KEY,

Name VARCHAR(50),

Price DECIMAL(10, 2) DEFAULT 0.00);

In this example, if the Price is not specified during an insert, it will default to 0.00.

**JOINS**

**Joins** are used to combine rows from two or more tables based on related columns.

**1. INNER JOIN**

* **Definition:** Retrieves records that have matching values in both tables.
* **Use Case:** When you need only the records that exist in both tables.
* **Example:**
* SELECT Employees.Name, Departments.DepartmentName
* FROM Employees

INNER JOIN Departments ON Employees.DepartmentID = Departments.ID;

**2. LEFT JOIN**

**LEFT JOIN (LEFT OUTER JOIN)**retrieves all records from the left table and matched records from the right table. Unmatched records from the right table are NULL.

* **Use Case:** When you want all records from the left table, regardless of matches in the right table.
* **Example:**
* SELECT Employees.Name, Departments.DepartmentName
* FROM Employees

LEFT JOIN Departments ON Employees.DepartmentID = Departments.ID;

**3. RIGHT JOIN**

**RIGHT JOIN (RIGHT OUTER JOIN)**retrieves all records from the right table and matched records from the left table. Unmatched records from the left table are NULL.

* **Use Case:** When you want all records from the right table, regardless of matches in the left table.
* **Example:**
* SELECT Employees.Name, Departments.DepartmentName
* FROM Employees

RIGHT JOIN Departments ON Employees.DepartmentID = Departments.ID;

**4. FULL OUTER JOIN**

**FULL OUTER JOIN**retrieves all records when there is a match in either left or right table. Unmatched records will have NULL values.

* **Use Case:** When you need all records from both tables, with matches where available.
* **Example:**
* SELECT Employees.Name, Departments.DepartmentName
* FROM Employees

FULL OUTER JOIN Departments ON Employees.DepartmentID = Departments.ID;

**5. SELF JOIN**

**SELF JOIN**joins a table with itself to compare rows within the same table.

* **Use Case:** Useful for hierarchical data, such as finding employees and their managers within the same table.
* **Example:**
* SELECT A.Name AS Employee, B.Name AS Manager
* FROM Employees A

INNER JOIN Employees B ON A.ManagerID = B.ID;

**SUBQUERIES**

are nested queries within another SQL query, used to perform complex filtering and data retrieval.

**1. Non-Correlated Subquery**

* **Definition:** A subquery that operates independently of the outer query.
* **Characteristics:**
  + Can be executed on its own.
  + Provides a static result set to the outer query.
* **Example:**

SELECT Name FROM Employees

WHERE DepartmentID IN (SELECT ID FROM Departments WHERE Location = 'New York');

**2. Correlated Subquery**

**Definition:** A subquery that depends on the outer query for its values and cannot run independently.

**Characteristics:**

Executes once for each row processed by the outer query.

Often used for row-by-row comparison.

**Example:**

SELECT E1.Name FROM Employees E1 WHERE E1.Salary > (SELECT AVG(E2.Salary) FROM Employees E2WHERE E2.DepartmentID = E1.DepartmentID);

**Aggregate Functions**

**Aggregate functions** perform calculations on multiple rows of a table's column and return a single value.

**1. SUM**

**Definition:** Calculates the total sum of a numeric column.

**Use Case:** To find the total sales, total expenses, etc.

**Example:**

SELECT SUM(Salary) AS TotalSalaries FROM Employees;

**2. COUNT**

**Definition:** Counts the number of rows that meet a specified condition.

**Use Case:** To determine the number of employees, orders, etc.

**Example:**

SELECT COUNT(\*) AS EmployeeCount FROM Employees;

**3. AVG (Average)**

**Definition:** Calculates the average value of a numeric column.

**Use Case:** To find the average salary, average score, etc.

**Example:**

SELECT AVG(Salary) AS AverageSalary FROM Employees;

**4. MAX**

**Definition:** Finds the highest value in a column.

**Use Case:** To identify the highest salary, latest hire date, etc.

**Example:**

SELECT MAX(Salary) AS HighestSalary FROM Employees;

**5.MIN**

**Definition:** Finds the lowest value in a column.

**Use Case:** To identify the lowest salary, earliest hire date, etc.

**Example:**

SELECT MIN(Salary) AS LowestSalary FROM Employees;

**CLAUSES**

 These clauses are used to group rows that have the same values in specified columns and to filter groups based on conditions.

**1. GROUP BY Clause**

**Definition:** Groups rows that have the same values in specified columns into summary rows.

**Use Case:** To aggregate data, such as total sales per department.

**Example:**

SELECT DepartmentID, COUNT(\*) AS EmployeeCount FROM Employees

GROUP BY DepartmentID;

**2. HAVING Clause**

**Definition:** Filters groups based on a specified condition, similar to WHERE but applied to aggregated data.

**Use Case:** To show only departments with more than 5 employees.

**Example:**

SELECT DepartmentID, COUNT(\*) AS EmployeeCount FROM Employees

GROUP BY DepartmentID

HAVING COUNT(\*) > 5;

**SET OPERATIONS**

**Set operations** combine the results of two or more SELECT statements into a single result set.

**1. UNION**

**Definition:** Combines the result sets of two SELECT statements and removes duplicate rows.

**Use Case:** To list all unique cities from customers and suppliers.

**Example:**

SELECT City FROM Customers

UNION SELECT City FROM Suppliers;

**2. INTERSECT**

**Definition:** Returns only the records that are common to both SELECT statements.

**Use Case:** To find cities that are both customer and supplier locations.

**Example:**

SELECT City FROM Customers

INTERSECT SELECT City FROM Suppliers;

**3. EXCEPT**

**Definition:** Returns records from the first SELECT statement that are not in the second SELECT statement.

**Use Case:** To list cities that have customers but no suppliers.

**Example:**

SELECT City FROM Customers

EXCEPT SELECT City FROM Suppliers;

**VIEWS**

**Definition:** A virtual table based on the result set of a SELECT statement.

**Benefits:**

Simplifies complex queries.

Enhances security by restricting data access.

Provides a consistent data representation.

**Example:**

CREATE VIEW ActiveEmployees AS

SELECT Name, DepartmentID, Salary

FROM Employees

WHERE IsActive = TRUE;

**INDEXES**

**Definition:** Database objects that improve the speed of data retrieval operations on a table.

**Benefits:**

Enhances query performance.

Speeds up searching and sorting operations.

**Considerations:**

Require additional storage.

May slow down INSERT, UPDATE, and DELETE operations.

**Example:**

CREATE INDEX idx\_department ON Employees (DepartmentID);

**STORED PROCEDURE**

**Definition:** Precompiled collections of SQL statements that perform specific tasks.

**Benefits:**

Promote code reuse.

Enhance performance by reducing parsing time.

Improve security by controlling data access.

**Example:**

CREATE PROCEDURE GetEmployeesByDept

@DeptID INT

AS

BEGIN

SELECT Name, Salary FROM Employees

WHERE DepartmentID = @DeptID;

END;

**FUNCTIONS**

**Definition:** SQL routines that return a single value or a table and can be used within SQL statements.

**Benefits:**

Encapsulate reusable logic.

Simplify complex calculations.

**Types:**

**Scalar Functions:** Return a single value.

**Table-Valued Functions:** Return a table.

**Example:**

CREATE FUNCTION GetEmployeeCount (@DeptID INT)

RETURNS INT

AS

BEGIN

DECLARE @Count INT;

SELECT @Count = COUNT(\*) FROM Employees WHERE DepartmentID = @DeptID;

RETURN @Count;

END;

**TRIGGERS**

**Definition:** Automated procedures that execute in response to specific events on a table, such as INSERT, UPDATE, or DELETE.

**Use Case:** To maintain audit logs or enforce business rules automatically.

**Example:**

CREATE TRIGGER trg\_AfterInsert

ON Employees

AFTER INSERT

AS

BEGIN

INSERT INTO AuditLog (Action, ActionDate)

VALUES ('New employee added', GETDATE());

END;

**EVENTS**

**Definition:** Scheduled tasks that execute SQL code at predefined times or intervals.

**Use Case:** To perform routine maintenance tasks, such as data cleanup or report generation.

**Example (MySQL Syntax):**

CREATE EVENT ev\_DailyCleanup

ON SCHEDULE EVERY 1 DAY

DO

DELETE FROM TempData WHERE CreatedDate < NOW() - INTERVAL 30 DAY;

**CURSORS**

**Cursors** are database objects used to retrieve, manipulate, and navigate through a result set one row at a time.

**Purpose:** To handle row-by-row processing in SQL, which is useful for operations that requireiterative logic.

**Components:**

**DECLARE:** Defines the cursor and the SELECT statement.

**OPEN:** Executes the SELECT statement and populates the cursor.

**FETCH:** Retrieves the next row from the cursor.

**CLOSE:** Closes the cursor.

**DEALLOCATE:** Frees the cursor resources.

**SQL OPERATORS**

The operators are symbols (and keywords) that are used to perform operations with values.

These operators are used with SQL clauses such as: SELECT, WHERE, ON etc.

The operators in SQL can be categorized as:

* Arithmetic operators
* Comparison operators
* Logical operators