SQL

In SQL, there are several types of commands that can be used to manipulate data and the database schema. The two main types of SQL commands are Data Definition Language (DDL) and Data Manipulation Language (DML). Here's a breakdown of what each of these types of commands do:

1. **Data Definition Language (DDL):** DDL commands are used to define the database schema, which includes creating and modifying tables, views, indexes, and other database objects. Some examples of DDL commands include:

   CREATE: Used to create tables, views, indexes, and other database objects.

   ALTER: Used to modify the structure of existing tables or other database objects.

   DROP: Used to delete tables, views, indexes, and other database objects.

2. **Data Manipulation Language (DML):** DML commands are used to manipulate data within the database. Some examples of DML commands include:

   INSERT: Used to insert data into a table.

   SELECT: Used to retrieve data from one or more tables.

   UPDATE: Used to modify data in a table.

   DELETE: Used to delete data from a table.

3. **Data Control Language (DCL):** DCL commands are used to control access to the database. Some examples of DCL commands include:

   GRANT: Used to grant privileges to a user or role.

   REVOKE: Used to revoke privileges from a user or role.

4. **Transaction Control Language (TCL):** TCL commands are used to manage transactions within the database. Some examples of TCL commands include:

   COMMIT: Used to save changes made within a transaction.

ROLLBACK: Used to undo changes made within a transaction.

In addition to the four main types of SQL commands **(DDL, DML, DCL, and TCL),** there are some other types of SQL operations that are commonly used in database management and analysis. These include:

1. **Aggregate functions**: These are functions used to perform calculations on a set of values and return a single value. Examples include SUM, AVG, MAX, MIN, and COUNT.

Syntax for SUM:

```
SELECT SUM(column_name)
FROM table_name;
```

Example:

```
SELECT SUM(sales)
FROM orders;
```

Example: Retrieve the total sales amount for all orders in the orders table.

Syntax for AVG:

```
SELECT AVG(quantity)
FROM order_details;
```

2. **Joins**: A join is used to combine data from two or more tables based on a related column. There are several types of joins, including INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN.

I) Inner Join: The INNER JOIN returns only the matching records between two tables.

Syntax for INNER JOIN:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
```

```
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT customers.customer_name, orders.order_date
FROM customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id;
```

II) Left Join:

The LEFT JOIN returns all records from the left table (i.e., the table specified first in the query), and matching records from the right table. If there is no matching record in the right table, the result will contain NULL values for the columns from the right table.

Syntax for LEFT JOIN:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Example:

```
SELECT customers.customer_name, orders.order_date
FROM customers
LEFT JOIN orders
ON customers.customer_id = orders.customer_id;
```

III) Right Join:

The RIGHT JOIN is similar to the LEFT JOIN, but it returns all records from the right table and matching records from the left table. If there is no matching record in the left table, the result will contain NULL values for the columns from the left table.

Syntax for RIGHT JOIN:

```
SELECT customers.customer_name, orders.order_date
FROM customers
RIGHT JOIN orders
ON customers.customer_id = orders.customer_id;
```

IV) FULL OUTER JOIN:

Syntax for FULL OUTER JOIN:

```
SELECT customers.customer_name, orders.order_date
FROM customers
FULL OUTER JOIN orders
ON customers.customer_id = orders.customer_id;
```

3. **Subqueries**: A subquery is a query that is embedded within another query. It is used to retrieve data from one table based on values from another table.

Syntax:

```
SELECT product_name, unit_price
FROM products
WHERE category_id IN (
SELECT category_id
FROM categories
WHERE category_name = 'Beverages'
);
```

Example: Retrieve the name and unit price of all products in the products table that belong to the 'Beverages' category.

4. **Views**: A view is a virtual table that is created based on a query. It allows you to simplify complex queries by creating a separate view that retrieves the data you need.

    Syntax for creating a view:

    CREATE VIEW customer_orders AS

    SELECT customers.customer_name, orders.order_date, order_details.product_name

    FROM customers

    INNER JOIN orders ON customers.customer_id = orders.customer_id

    INNER JOIN order_details ON orders.order_id = order_details.order_id;

    Example: Create a view called customer_orders that combines information from the customers, orders, and order_details tables.

    Syntax for selecting from a view:

    SELECT customer_name, order_date, product_name

    FROM customer_orders;

    Example: Retrieve the customer name, order date, and product name for all orders in the customer_orders view.

5. **Indexes**: An index is a data structure that improves the speed of data retrieval operations on a table. It allows you to quickly search for specific values or ranges of values in a table.

    Syntax:

    CREATE INDEX order_customer_idx

    ON orders (customer_id);

    Example:

    Create an index called order_customer_idx on the customer_id column in the orders table.

Stored procedures: A stored procedure is a set of SQL statements that is stored in the database and can be executed by a user or application. It is useful for performing complex data manipulation operations or for automating repetitive tasks.

Syntax:

```
CREATE PROCEDURE get_customer_orders
@customer_name VARCHAR(50)
AS
BEGIN
SELECT customers.customer_name, orders.order_date, order_details.product_name
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id
INNER JOIN order_details ON orders.order_id = order_details.order_id
WHERE customers.customer_name = @customer_name;
END;
```

Example: Create a stored procedure called get_customer_orders that retrieves the order date and product name for all orders placed by a specific customer.

Syntax for executing a stored procedure:

```
EXEC get_customer_orders 'Acme Inc.';
```

Example: Execute the get_customer_orders stored procedure for the customer 'Acme Inc.' to retrieve their order information.

Basic flow for SQL:

1.**Create a table**: Use the CREATE TABLE command to create a new table. For example, you can create a table called "employees" with columns for employee ID, name, and salary using the following SQL code:

Before creating table, ensure:

-- Create a new database

CREATE DATABASE my_database;

-- Use the new database

USE my_database;

Syntax:

CREATE TABLE employees (

employee_id INT,

name VARCHAR(50),

salary INT

);

2. **Insert data:** Use the INSERT INTO command to add data to the table. For example, you can add a row for an employee named John Doe with an employee ID of 123 and a salary of 50000 using the following SQL code:

Syntax:

-- Inserting multiple values

INSERT INTO employees (employee_id, name, salary) VALUES

(123, 'John Doe', 50000),

(456, 'Peter Parker', 70000),

(123, 'Jacob', 90000),

(123, 'John Parker', 10000),

3. **Select data:** Use the SELECT command to retrieve data from the table. For example, you can select all rows from the employees table using the following SQL code:

Syntax:

SELECT * FROM employees;

4. **Filter data:** Use the WHERE clause to filter the results of a SELECT command. For example, you can select only the rows where the employee ID is 123 using the following SQL code:

Syntax:

SELECT * FROM employees WHERE employee_id = 123;

5. **Update data**: Use the UPDATE command to modify data in the table. For example, you can update John Doe's salary to 60000 using the following SQL code:

Syntax:

UPDATE employees SET salary = 60000 WHERE employee_id = 123;

6. **Delete data:** Use the DELETE command to remove data from the table. For example, you can delete the row for John Doe using the following SQL code:

Syntax:

DELETE FROM employees WHERE employee_id = 123;