Some of The Most Important SQL Commands

- `SELECT` - extracts data from a database
- `UPDATE` - updates data in a database
- `DELETE` - deletes data from a database
- `INSERT INTO` - inserts new data into a database
- `CREATE DATABASE` - creates a new database
- `ALTER DATABASE` - modifies a database
- `CREATE TABLE` - creates a new table
- `ALTER TABLE` - modifies a table
- `DROP TABLE` - deletes a table
- `CREATE INDEX` - creates an index (search key)
- `DROP INDEX` - deletes an index

SELECT

```
SELECT CustomerName, City FROM Customers;
```

```
SELECT column1, column2, ...
FROM table_name;
```

Return all the columns from the Customers table:`SELECT * FROM Customers;`

SELECT DISTINCT

```
SELECT DISTINCT Country FROM Customers;
```

```
SYNTAX: SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Count Distinct

By using the `DISTINCT` keyword in a function called `COUNT`, we can return the number of different countries.

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

WHERE: The WHERE clause is used to filter records.It is used to extract only those records that fulfill a specified condition.

```
SELECT * FROM Customers   WHERE Country='Mexico';
```

```
SELECT column1, column2, …  FROM table_name  WHERE condition;
```

Example:

```
SELECT * FROM Customers  WHERE CustomerID > 80;
```

The SQL ORDER BY

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

Sort the products by price: `SELECT * FROM Products  ORDER BY Price;`

Syntax:

```
SELECT column1, column2, …  FROM table_name  ORDER BY column1, column2, ... ASC|DESC;
```

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Example: Sort the products from highest to lowest price:

```
SELECT * FROM Products ORDER BY Price DESC;
```

Order Alphabetically :

For string values the ORDER BY keyword will order alphabetically

Example: Sort the products alphabetically by ProductName:

```
SELECT * FROM Products ORDER BY ProductName;
```

Sort the products by ProductName in reverse order:

```
SELECT * FROM Products ORDER BY ProductName DESC;
```

Example

```
SELECT * FROM Customers ORDER BY Country ASC, CustomerName DESC;
```

The SQL AND Operator

The `WHERE` clause can contain one or many `AND` operators.

The `AND` operator is used to filter records based on more than one condition, like if you want to return all customers from Spain that starts with the letter 'G':`SELECT * FROM Customers  WHERE Country = 'Spain' AND CustomerName LIKE 'G%';`

Syntax:`SELECT column1, column2, …  FROM table_name WHERE condition1 AND condition2 AND condition3 ...;`

Example:`SELECT * FROM Customers  WHERE Country = 'Germany' AND City = 'Berlin' AND PostalCode > 12000;`

Combining AND and OR:Select all Spanish customers that starts with either "G" or "R"

```
SELECT * FROM Customers WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');
```

OR Syntax:`SELECT column1, column2, …  FROM table_name WHERE condition1 OR condition2 OR condition3 ...;`

The NOT Operator:The `NOT` operator is used in combination with other operators to give the opposite result, also called the negative result.

Select only the customers that are NOT from Spain: `SELECT * FROM Customers WHERE NOT Country = 'Spain';`

NOT LIKE Example:Select customers that does not start with the letter 'A': `SELECT * FROM Customers WHERE CustomerName NOT LIKE 'A%';`

NOT BETWEEN:Example:Select customers with a customerID not between 10 and 60: `SELECT * FROM Customers WHERE CustomerID NOT BETWEEN 10 AND 60;`

```
SELECT * FROM Customers
WHERE City NOT IN ('Paris', 'London');
```

INSERT INTO Syntax:
```
INSERT INTO table_name
(column1,column2, column3, )VALUES (value1, value2, value3,.);
```

IS NULL Syntax:
```
SELECT column_names FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax:
```
SELECT column_names FROM table_name
WHERE column_name IS NOT NULL;
```

UPDATE Syntax:
```
UPDATE table_name SET column1 = value1,
column2 = value2, … WHERE condition;
```

Note: Be careful when updating records in a table! Notice the `WHERE` clause in the `UPDATE` statement. The `WHERE` clause specifies which record(s) that should be updated. If you omit the `WHERE` clause, all records in the table will be updated!

DELETE Syntax:
```
DELETE FROM table_name WHERE condition;
```

The SQL SELECT TOP Clause

```
SELECT TOP 3 * FROM Customers;
```

Not all database systems support the `SELECT TOP` clause. MySQL supports the `LIMIT` clause to select a limited number of records, while Oracle uses `FETCH FIRST n ROWS ONLY` and `ROWNUM`.

MySQL Syntax: `SELECT column_name(s) FROM table_name WHERE condition LIMIT number;`

The SQL MIN() and MAX() Functions

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

Find the lowest price: `SELECT MIN(column name) FROM table name;`

Find the highest price: `SELECT MAX(column name) FROM table name;`

```
SELECT MIN(Price) AS SmallestPrice FROM Products;
```

The SQL COUNT() Function:The `COUNT()` function returns the number of rows that matches a specified criterion.

```
SELECT COUNT(column_name) FROM table_name WHERE condition;
```

Example:Find the number of products where `Price` is higher than 20:

```
SELECT COUNT(ProductID) FROM Products WHERE Price > 20;
```

Ignore Duplicates:How many *different* prices are there in the `Products` table:

```
SELECT COUNT(DISTINCT Price) FROM Products;
```

```
SELECT COUNT(*) AS [number of records] FROM Products;
```

The SQL SUM() Function:The `SUM()` function returns the total sum of a numeric column.

Syntax:

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

```
SELECT SUM(Quantity) AS total
FROM OrderDetails;
```

Use an expression inside the `SUM()` function:

```
SELECT SUM(Quantity * 10) FROM OrderDetails;
```

The SQL AVG() Function

The `AVG()` function returns the average value of a numeric column.

```
SELECT AVG(column_name) FROM table_name WHERE condition;
```

```
SELECT AVG(Price) AS [average price]  FROM Products;
```

Return all products with a higher price than the average price:

```
SELECT * FROM Products WHERE price > (SELECT AVG(price) FROM
Products);
```

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.There are two wildcards often used in conjunction with the LIKE operator:1. The percent sign % represents zero, one, or multiple characters

2. The underscore sign _ represents one, single character

Select all customers that starts with the letter "a":

```
SELECT * FROM Customers WHERE CustomerName LIKE 'a%';
```

```
SELECT column1,column2,FROM table_name WHERE columnN LIKE
pattern;
```

Return all customers that starts with "b" and ends with "s":

```
SELECT * FROM Customers WHERE CustomerName LIKE 'b%s';
```

SQL Wildcard Characters: A wildcard character is used to substitute one or more characters in a string.Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

The SQL IN Operator:The IN operator allows you to specify multiple values in a WHERE clause.The IN operator is a shorthand for multiple OR conditions.

Return all customers from 'Germany', 'France', or 'UK' `SELECT * FROM Customers  WHERE Country IN ('Germany', 'France', 'UK');`

```
SELECT column_name(s)  FROM table_name WHERE column_name IN
(value1, value2, ...);
```

Example: Return all customers that have an order in the Orders table:

```
SELECT * FROM Customers WHERE CustomerID IN (SELECT CustomerID
FROM Orders);
```

Return all customers that have NOT placed any orders in the Orders table:

```
SELECT * FROM Customers

WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);
```

The SQL BETWEEN Operator:The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.The BETWEEN operator is inclusive: begin and end values are included.

Selects all products with a price between 10 and 20: `SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;`

```
SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;
```

Example :`SELECT * FROM Products WHERE Price NOT BETWEEN 10 AND 20;`

BETWEEN with IN:The following SQL statement selects all products with a price between 10 and 20. In addition, the CategoryID must be either 1,2, or 3:

Example:`SELECT * FROM Products WHERE Price BETWEEN 10 AND 20 AND CategoryID IN (1,2,3);`

```
SELECT * FROM Orders WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.Aliases are often used to make column names more readable.An alias only exists for the duration of that query.An alias is created with the `AS` keyword.

SELECT CustomerID AS ID FROM Customers;

SQL JOIN:A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

Example:`SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;`

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table

The SQL GROUP BY Statement

- The `GROUP BY` statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The `GROUP BY` statement is often used with aggregate functions (`COUNT()`, `MAX()`, `MIN()`, `SUM()`, `AVG()`) to group the result-set by one or more columns.
- GROUP BY Syntax:
```
SELECT column_name(s) FROM table_name
WHERE condition GROUP BY column_name(s) ORDER BY
column_name(s);
```
- The following SQL statement lists the number of customers in each country, sorted high to low:
- Example
```
SELECT COUNT(CustomerID), Country FROM Customers
GROUP BY Country ORDER BY COUNT(CustomerID) DESC;
```

The SQL HAVING Clause

- The `HAVING` clause was added to SQL because the `WHERE` keyword cannot be used with aggregate functions.
- HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

I can't directly provide the entire syntax from W3Schools, but I can guide you through the various MySQL statements and commands. MySQL has numerous commands for database creation, manipulation, querying, user management, and more. Here are some fundamental ones:

- **Database Operations:**
  - `CREATE DATABASE`: Creates a new database.
  - `DROP DATABASE`: Deletes an existing database.
  - `USE DATABASE`: Selects a specific database for use.

- **Table Operations:**
  - `CREATE TABLE`: Creates a new table in a database.
  - `ALTER TABLE`: Modifies an existing table's structure.
  - `DROP TABLE`: Deletes a table.

- **Data Manipulation:**
  - `SELECT`: Retrieves data from a database.
  - `INSERT INTO`: Adds new records into a table.
  - `UPDATE`: Modifies existing records in a table.
  - `DELETE FROM`: Removes records from a table.

- **Data Filtering and Sorting:**
  - `WHERE`: Filters records based on specified conditions.
  - `ORDER BY`: Sorts the result set in ascending or descending order.
  - `GROUP BY`: Groups rows that have the same values.

- **Joins:**
  - `INNER JOIN`: Returns records that have matching values in both tables.

  - `LEFT JOIN`: Returns all records from the left table and the matched records from the right table.
  - `RIGHT JOIN`: Returns all records from the right table and the matched records from the left table.
  - `FULL OUTER JOIN`: Returns all records when there is a match in either the left or right table.

- **Constraints:**
  - `PRIMARY KEY`: Uniquely identifies each record in a table.
  - `FOREIGN KEY`: Enforces referential integrity for a relationship between tables.
  - `UNIQUE`: Ensures unique values in a column.

- **User Management:**
  - `CREATE USER`: Creates a new user account.
  - `GRANT`: Assigns privileges to a user account.
  - `REVOKE`: Removes privileges from a user account.
  - `DROP USER`: Deletes a user account.

Each command has its own specific syntax and variations depending on the functionality needed. W3Schools offers detailed documentation and examples for each of these commands and more on their MySQL section.

Absolutely! Here are basic examples for some of the fundamental MySQL commands:

- **Database Operations:**

  - `CREATE DATABASE`:

    ```sql
    CREATE DATABASE my_database;
    ```

  - `DROP DATABASE`:

    ```sql
    DROP DATABASE my_database;
    ```

  - `USE DATABASE`:

```sql
USE my_database;
```

- **Table Operations:**

  - `CREATE TABLE`:
    ```sql
    CREATE TABLE users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        username VARCHAR(50) NOT NULL,
        email VARCHAR(100) NOT NULL UNIQUE
    );
    ```

  - `ALTER TABLE`:
    ```sql
    ALTER TABLE users
    ADD COLUMN age INT;
    ```

  - `DROP TABLE`:
    ```sql
    DROP TABLE users;
    ```

- **Data Manipulation:**

  - `SELECT`:

```sql
SELECT * FROM users;
```

- `INSERT INTO`:

```sql
INSERT INTO users (username, email) VALUES ('JohnDoe', 'john@example.com');
```

- `UPDATE`:

```sql
UPDATE users
SET age = 30
WHERE username = 'JohnDoe';
```

- `DELETE FROM`:

```sql
DELETE FROM users
WHERE id = 1;
```

- **Data Filtering and Sorting:**

- `WHERE`:

```sql
SELECT * FROM users
WHERE age > 25;
```

```

- `ORDER BY`:

  ```sql
  SELECT * FROM users
  ORDER BY username ASC;
  ```

- `GROUP BY`:

  ```sql
  SELECT department, COUNT(*) AS employee_count
  FROM employees
  GROUP BY department;
  ```

These examples iOf course, here are examples for the remaining MySQL commands:

- **Joins:**

  - `INNER JOIN`:

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

  - `LEFT JOIN`:

    ```sql

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

- `RIGHT JOIN`:

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

- `FULL OUTER JOIN` (MySQL doesn't directly support FULL OUTER JOIN, but you can simulate it):

```sql
SELECT *
FROM table1
LEFT JOIN table2 ON table1.column = table2.column
UNION
SELECT *
FROM table1
RIGHT JOIN table2 ON table1.column = table2.column;
```

- **Constraints:**

- `PRIMARY KEY`:

```sql
CREATE TABLE products (
```

```
    product_id INT PRIMARY KEY,
    product_name VARCHAR(50)
);
```

- `FOREIGN KEY`:
  ```sql
  CREATE TABLE orders (
      order_id INT PRIMARY KEY,
      product_id INT,
      FOREIGN KEY (product_id) REFERENCES products(product_id)
  );
  ```

- `UNIQUE`:
  ```sql
  CREATE TABLE employees (
      employee_id INT PRIMARY KEY,
      employee_email VARCHAR(100) UNIQUE
  );
  ```

- **User Management:**

  - `CREATE USER`:
    ```sql
    CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
    ```

- `GRANT`:

  ```sql
  GRANT SELECT, INSERT ON my_database.* TO 'newuser'@'localhost';
  ```

- `REVOKE`:

  ```sql
  REVOKE SELECT ON my_database.* FROM 'newuser'@'localhost';
  ```

- `DROP USER`:

  ```sql
  DROP USER 'newuser'@'localhost';
  ```

These examples showcase various MySQL commands for performing joins, setting constraints, and managing users within a database. They serve as a starting point for understanding and utilizing these functionalities in MySQL databases.llustrate the basic usage of some of the key MySQL commands. You can further explore and combine these commands to suit your specific database needs.