# **SQL Command Types :**

The operations performed in SQL can be categorized into five major commands , Each category serves a different purpose and contains commands to perform various operations:

* DDL : (Data Definition Language)
* DML : (Data Manipulation Language)
* DCL : (Data Control Language)
* TCL : (Transaction Control Language)
* DQL : (Data Query Language)

### **DDL : (Data Definition Language)**

* DDL commands are used to define, modify, and delete the structure of database objects such as tables, indexes, and constraints
* They don't manipulate data itself, but rather define the structure of how data is stored in the database.

#### **Example :**

* CREATE: Creates a new table or database.
* ALTER: Modifies the structure of an existing database object.
* DROP: Deletes an entire table or database.
* TRUNCATE: Deletes all records from a table without deleting the table itself.

### **DML : (Data Manipulation Language)**

* DML commands are used for managing data within schema objects.
* They are used to insert, update, or delete records from a database.

#### **Example :**

* INSERT: Adds new records to a table.
* UPDATE: Modifies existing records in a table.
* DELETE: Removes records from a table.

### **DCL : (Data Control Language)**

DCL commands are concerned with rights, permissions, and other controls of the database system.

#### **Examples :**

* GRANT: Gives user's access privileges to the database.
* REVOKE: Withdraws user's access privileges given by the GRANT command.

### **TCL : (Transaction Control Language)**

* TCL commands deal with the transaction operations within the database.
* These commands help in managing the changes made by DML statements.

#### **Examples :**

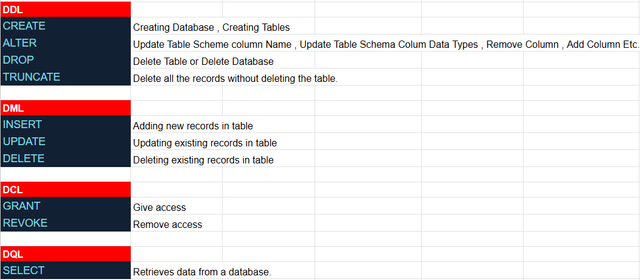
* COMMIT: Saves all the transactions to the database.
* ROLLBACK: Restores the database to the last committed state.
* SAVEPOINT: Sets a savepoint within a transaction.

### **DQL (Data Query Language)**

DQL is primarily used to fetch data from databases. It uses only one command:

#### **Examples :**

* SELECT: Retrieves data from a database.



# **DDL - ALTER (Altering Table details)**

### **Updating Column name in the table**

Syntax :

ALTER TABLE table\_name CHANGE COLUMN old\_column new\_column data\_types;

### **Updating data types or constraint for a column**

Syntax :

ALTER TABLE table\_name MODIFY COLUMN existing\_column\_name datatype constraint;

### **Adding a new column in a existing table**

Syntax :

ALTER TABLE table\_name

ADD COLUMN new\_column\_name data\_type constraints;

### **Removing a column from a existing table**

Syntax :

ALTER TABLE table\_name

DROP COLUMN column\_name;

# **DDL - DROP & TRUNCATE**

### **How to delete a Whole table**

It will delete the whole table along with all the data. Syntax :

DROP TABLE table\_name;

### **How to delete an existing database :**

Syntax :

DROP DATABASE database\_name;

### **TRUNCATE**

It is used to delete all the records from a table quickly and efficiently, without affecting the structure of the table.

Syntax :

TRUNCATE TABLE table\_name;

# **Arithmetic Operators in SQL**

Arithmetic operators in SQL are used to perform mathematical calculations on data stored in tables.

### **Arithmetic Operators With Select**

-- Addition

SELECT (column1 + column2) FROM tableName ;

-- Subtraction

SELECT (column1 - column2) FROM tableName

-- Multiplication

SELECT (column1 \* column2) FROM tableName

-- Division

SELECT (column1 / column2) FROM tableName

-- Modulus

SELECT (column1 % column2) FROM tableName

### **Arithmetic Operators With Where**

-- Addition

SELECT \* FROM tableName WHERE column1 + column2 = colum\_value;

-- Subtraction

SELECT \* FROM tableName WHERE column1 - column2 = column\_value;

-- Multiplication

SELECT \* FROM tableName WHERE column1 \* column2 < column\_value;

-- Division

SELECT \* FROM tableName WHERE column1 / column2 = column\_value;

-- Modulus

SELECT \* FROM tableName WHERE column1 % 2 = column\_value;

### **Example :**

Fetch 5

SELECT 5;

Fetch result of 5+5 and store the result in a column "su\_value"

SELECT 5 + 5 AS sum\_value;

Fetch result of 5-5 and store the result in a column "su\_value"

SELECT 5 - 5 AS sum\_value;

Calculate discount price for each product from bigbasketproducts

SELECT Product , (market\_price - sale\_price) AS Discount\_price FROM bigbasketproducts;

Calculate discount price for each product from bigbasketproducts, Give me only those products where discount is greater than 50.

SELECT Product , (market\_price - sale\_price) AS test

FROM bigbasketproducts

WHERE (market\_price - sale\_price) > 50 ;

-- We can write like this also if we dont need discount

SELECT \*

FROM bigbasketproducts

WHERE (market\_price - sale\_price) > 50 ;

# **Comparison Operators**

Comparison operators in SQL are fundamental tools that allow you to perform comparisons between values in your database, enabling you to filter, compare, and evaluate data based on specific criteria.

### **Equal**

SELECT \* FROM tableName WHERE column1 = 'value';

-- Example

SELECT \* FROM employees WHERE department = 'Sales';

### **Not Equal**

SELECT \* FROM tableName WHERE column1 != 'value';

-- OR

SELECT \* FROM tableName WHERE column1 <> 'value';

-- Example

SELECT \* FROM employees WHERE salary <> 50000;

### **Greater Than or Greater than Equal**

SELECT \* FROM tableName WHERE column1 > value;

SELECT \* FROM tableName WHERE column1 >= value;

-- Example

SELECT \* FROM employees WHERE age > 30;

### **Less Than or Less than Equal**

SELECT \* FROM tableName WHERE column1 < value;

SELECT \* FROM tableName WHERE column1 <= value;

-- Example

SELECT \* FROM products WHERE price < 100;

### **BETWEEN**

The BETWEEN operator is used to filter the result set within a certain range. The values can be numbers, text, or dates. This is inclusive of the boundary values.

SELECT \* FROM table\_Name WHERE column\_name BETWEEN smallValue AND largeValue;

-- Examples

SELECT \* FROM products WHERE price BETWEEN 50 AND 100;

### **IS NULL**

The IS NULL operator is used to test for empty values (NULL values).

SELECT \* FROM tableName WHERE column1 IS NULL;

-- Example

SELECT \* FROM employees WHERE commission IS NULL;

### **IS NOT NULL**

The IS NOT NULL operator is used to test for non-empty values.

SELECT \* FROM tableName WHERE column1 IS NOT NULL;

-- Example

SELECT \* FROM employees WHERE commission IS NOT NULL;

### **Example**

Give all the details of Rich People Where country\_of\_residence is India

SELECT \* FROM richperson

WHERE country\_of\_residence = "India";

Give all the rich people where age is less than 30

SELECT \* FROM richperson

WHERE age < 30;

Give all the rich people where age is less than equal to 30 and greater than equal to 18

SELECT \* FROM richperson

WHERE age BETWEEN 18 AND 29;

Give all the rich people where column wealth is null

SELECT \* FROM richperson

WHERE wealth IS NULL;

Give all the anme Rich people where column wealth is not null

SELECT full\_name FROM richperson

WHERE wealth IS NOT NULL;

# **Logical Operators**

### **AND**

The AND operator is used when you want to retrieve records that satisfy all the conditions specified.

SELECT \* FROM tableName WHERE column1 = 'value' AND column2 = 'value';

-- Example

SELECT \* FROM employees WHERE department = 'Sales' AND salary > 50000;

### **OR**

The OR operator is used to fetch records that satisfy at least one of the given conditions.

SELECT \* FROM tableName WHERE column1 = 'value' OR column2 = 'value';

-- Examples

SELECT \* FROM employees WHERE department = 'Sales' OR department = 'Marketing';

### **NOT**

The NOT operator negates a condition, selecting records if the condition is false.

SELECT \* FROM tableName WHERE NOT (column1 = 'value');

-- Example

SELECT \* FROM employees WHERE NOT department = 'Sales';

### **IN**

Allows you to specify a list of values for a column in your condition. It is , essentially a shorthand for multiple OR conditions.

SELECT \* FROM table\_Name WHERE column NOT IN ('value\_1', 'value\_2', 'value\_3');

-- Examples

SELECT \* FROM employees WHERE department IN ('Sales', 'Marketing');

### **Combining Logical Operators**

Logical operators can be combined in a single SQL statement to form complex conditions. The order of operations (precedence) is important in such cases, with NOT evaluated first, followed by AND, and then OR. Parentheses can be used to override this order and group conditions logically.

### **Example**

Give all the data where Brand is "OPPO" and Original Price is greater than 15000.

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000;

Give all the data where Brand is either "OPPO" or "ASUS"

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" OR Brand = "ASUS";

SELECT \* FROM flipkart\_mobiles

WHERE Brand IN ( "OPPO" , "ASUS");

Give all the data Where Color is not "Sky White"

SELECT \* FROM flipkart\_mobiles

WHERE NOT Color = "flipkart\_mobiles" AND NOT Color = "Moonlight Black" ;

Give all the data where Brand is "OPPO" and Original Price is greater than 15000 without the colour - Mint Cream

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000 AND NOT (Color = "Mint Cream");

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000 AND Color <> "Mint Cream";

Give all the data where Brand is "OPPO" and Original Price is greater than 15000 without the colour - Mint Cream and either rating is greater than 4.5 or the discount is greater than 5000. Sort the final data based on ascending order of Selling price

SELECT \* , (`Original Price` - `Selling Price`) FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000 AND Color <> "Mint Cream"

AND (Rating > 4.5 OR (`Original Price` - `Selling Price`) > 5000)

ORDER BY `Selling Price` ASC;

# **Sorting**

* Sorting in MySQL is primarily accomplished through the ORDER BY clause.
* This powerful feature allows you to sort the results of your SELECT queries based on one or more columns, in ascending or descending order.

### **Ascending Order**

Specifies an ascending order (this is the default). Rows will be sorted from lowest to highest.

SELECT \* FROM table\_name ORDER BY column\_name ASC;

### **Descending Order**

Specifies a descending order. Rows will be sorted from highest to lowest.

SELECT \* FROM table\_name ORDER BY column\_name DESC;

### **Multiple Sorting**

You can also sort by multiple columns. This is particularly useful when you have rows with the same value in the primary sort column and want to further define how those rows are sorted based on additional columns.

SELECT \* FROM table\_name ORDER BY column\_1 ASC , col\_2 DESC;

### **Example**

Sort data based on Salary in ascending order (Low -> High).

SELECT \* FROM emp

ORDER BY salary;

Sort data based on Salary in descending order (High -> Low).

SELECT \* FROM emp

ORDER BY salary DESC;

Sort the data based on emp name in reverse alphabetical order

SELECT \* FROM emp

ORDER BY name DESC;

Sort the data based on Ascending order of salary , if salary comes same for both employee then sort in alphabetical order of name

SELECT \* FROM emp

ORDER BY salary ASC , name ASC;

# **LIMIT and OFFSET**

### **LIMIT Clause**

The LIMIT clause restricts the number of rows returned in a query result. It can be especially useful in situations where you only need a subset of the rows from a larger dataset, such as retrieving the top N records.

Syntax

SELECT column\_names FROM table\_name LIMIT number\_of\_rows;

Example Give first 10 data from employees.

SELECT \* FROM employees LIMIT 10;

Give first 25 data from employees.

SELECT \* FROM employees LIMIT 25;

Give all the data where Brand is "OPPO" and Original Price is greater than 15000 without the colour - Mint Cream and either rating is greater than 4.5 or the discount is greater than 5000. Sort the final data based on ascending order of Selling price , Give such top 5 Products.

SELECT \* , (`Original Price` - `Selling Price`) FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000 AND Color <> "Mint Cream"

AND (Rating > 4.5 OR (`Original Price` - `Selling Price`) > 5000)

ORDER BY `Selling Price` ASC LIMIT 5;

### **OFFSET Clause**

The OFFSET clause is used to specify from which row to start retrieving data. This is useful for pagination, where you might want to skip a specific number of rows and fetch the next set of rows from the database.

Syntax when used with LIMIT:

SELECT column\_names FROM table\_name LIMIT number\_of\_rows OFFSET start\_row;

Example: This query skips the first 20 rows and returns the next 10 rows from the employees table.

SELECT \* FROM employees LIMIT 10 OFFSET 20;

### **Pagination :**

Assuming you display 10 rows per page, for page 1:

SELECT \* FROM employees LIMIT 10;

For page 3 (skipping the first 20 rows to get rows 21-30):

SELECT \* FROM employees LIMIT 10 OFFSET 20;

Shorthand Syntax Example:

SELECT \* FROM employees LIMIT 20, 10;

# **Distinct**

* The DISTINCT keyword in MySQL is used to remove duplicate rows from a result set, returning only unique rows.
* It can be particularly useful when you have multiple records that are identical in one or several columns and you want to ensure that your query result includes each unique value only once.
* This feature is often used in queries involving the SELECT statement.

### **Syntax :**

Single Column:

SELECT DISTINCT column\_name FROM table\_name;

Multiple Columns:

SELECT DISTINCT column1, column2 FROM table;

### **Example :**

Give all the distinct Brands who sales Mobile

SELECT DISTINCT Brand FROM flipkart\_mobiles;

Give all the distinct Brands and Color from Mobile database

SELECT DISTINCT Brand , Color FROM flipkart\_mobiles;

Give all the Unique Director who produces movie in Netflix.

SELECT DISTINCT director FROM netflix;

Give all the unique director name who produces movie in Nexflix and belongs to the country India

SELECT DISTINCT director , country FROM netflix WHERE country = "India";

<https://drive.google.com/drive/folders/14HZncSGcRGWFkLMNm9eD5uh_F6RSz18M?usp=drive_link>

-- Use database db101

USE db101;

-- Show all the tables inside db101

SHOW TABLES;

-- Create a new table productDetails with the specified fields

CREATE TABLE productDetails(

productID INT PRIMARY KEY,

productName VARCHAR(30),

category TEXT,

price DECIMAL(10,2),

quantityInStock INT,

dateAdded DATE

);

-- Insert a record into productDetails

INSERT INTO productDetails VALUES (1, 'mobile', 'elec', 200.20, 1, '2023-10-12');

-- Select all records from productDetails

SELECT \* FROM productDetails;

-- Show all the tables inside db101

SHOW TABLES;

-- Describe the structure of productDetails

DESC productDetails;

-- Describe the structure of productDetails (alternative syntax)

DESCRIBE productDetails;

-- Update the column name "quantityInStock" to "quantity"

ALTER TABLE productDetails CHANGE COLUMN quantityInStock quantity INT;

-- Describe the structure of productDetails after renaming the column

DESCRIBE productDetails;

-- Select all records from productDetails after renaming the column

SELECT \* FROM productDetails;

-- Update the data type of the column "category" to VARCHAR(25)

ALTER TABLE productDetails MODIFY COLUMN category VARCHAR(25);

-- Describe the structure of productDetails after modifying the column

DESCRIBE productDetails;

-- Add a new column "isBestSeller" of type BOOLEAN to productDetails

ALTER TABLE productDetails ADD COLUMN isBestSeller BOOLEAN;

-- Add a new column "descp" of type TEXT to productDetails

ALTER TABLE productDetails ADD COLUMN descp TEXT;

-- Describe the structure of productDetails after adding new columns

DESCRIBE productDetails;

-- Select all records from productDetails after adding new columns

SELECT \* FROM productDetails;

-- Remove the column "isBestSeller" from productDetails

ALTER TABLE productDetails DROP COLUMN isBestSeller;

-- Describe the structure of productDetails after removing the column

DESCRIBE productDetails;

-- Drop the entire table productDetails

DROP TABLE productDetails;

-- Truncate the table productDetails (this will fail because the table was dropped)

TRUNCATE TABLE productDetails;

-- Describe the structure of productDetails (this will fail because the table was dropped)

DESCRIBE productDetails;

-- Select all records from productDetails (this will fail because the table was dropped)

SELECT \* FROM productDetails;

-- Drop an existing database (replace "database\_name" with the actual database name)

DROP DATABASE database\_name;

-- Truncate command explanation

-- It is used to delete all the records from a table quickly and efficiently, without affecting the structure of the table.

-- You have 10 records in your table and you want to delete 1 record with id 1

DELETE FROM productDetails WHERE productID = 1;

-- 1. Give all the results from the table - bigbasket

SELECT \* FROM bigbasket;

-- 2. Can you print 5 in sql ?

SELECT 5;

-- 3. Print result of 5+5 and store the result in a column "sum\_value" (Alias)

SELECT 5 + 5 AS sum\_value;

-- 4. Fetch result of 5-5 and store the result in a column "su\_value"

SELECT 5 - 5 AS sum\_value;

-- Similarly you can do for - / , % , \*

-- 5. Calculate discount price [market\_price - sale\_price] for each product from bigbasketproducts

SELECT Product , (market\_price - sale\_price) AS Discount\_price FROM bigbasket;

-- 6. Calculate discount price for each product from bigbasketproducts, Give me only those products where discount is greater than 50.

SELECT Product , (market\_price - sale\_price) AS Discount\_price

FROM bigbasket

WHERE (market\_price - sale\_price) > 50 ;

-- 7. We can write like this also if we dont need discount

SELECT \*

FROM bigbasket

WHERE (market\_price - sale\_price) > 50 ;

-- Comparison

-- 8. Give all the details of Rich People Where country\_of\_residence is India

SELECT \* FROM richperson

WHERE country\_of\_residence = "India";

-- 9. Give all the rich people where age is less than 30

SELECT \* FROM richperson

WHERE age < 30;

-- 10. Give all the rich people where age is less than equal to 30 and greater than equal to 18

SELECT \* FROM richperson

WHERE age BETWEEN 18 AND 29;

-- 11. Give all the rich people where column wealth is null

SELECT \* FROM richperson

WHERE wealth IS NULL;

-- 12. Give all the anme Rich people where column wealth is not null

SELECT full\_name FROM richperson

WHERE wealth IS NOT NULL;

-- Logical Operators

-- 13. Give all the data where Brand is "OPPO" and Original Price is greater than 15000.

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000;

-- 14. Give all the data where Brand is either "OPPO" or "ASUS"

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" OR Brand = "ASUS";

SELECT \* FROM flipkart\_mobiles

WHERE Brand IN ( "OPPO" , "ASUS");

-- 15. Give all the data Where Color is not "Sky White"

SELECT \* FROM flipkart\_mobiles

WHERE NOT Color = "flipkart\_mobiles" AND NOT Color = "Moonlight Black" ;

-- 16. Give all the data where Brand is "OPPO" and Original Price is greater than 15000 without the colour - Mint Cream

-- Brand = "OPPO"

-- `Original Price` > 15000

-- Color = "Mint Cream" (Not)

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000 AND NOT (Color = "Mint Cream");

SELECT \* FROM flipkart\_mobiles

WHERE Brand = "OPPO" AND `Original Price` > 15000 AND Color <> "Mint Cream";

-- Sorting

-- Sort data based on Salary in ascending order (Low -> High).

SELECT \* FROM emp

ORDER BY salary;

-- Sort data based on Salary in descending order (High -> Low).

SELECT \* FROM emp

ORDER BY salary DESC;

-- Sort the data based on emp name in reverse alphabetical order

SELECT \* FROM emp

ORDER BY name DESC;

-- Sort the data based on Ascending order of salary , if salary comes same for both employee then sort in alphabetical order of name

SELECT \* FROM emp

ORDER BY salary ASC , name ASC;

-- Give first 10 data from employees.

SELECT \* FROM employees LIMIT 10;

-- Give first 25 data from employees.

SELECT \* FROM employees LIMIT 25;

-- This query skips the first 20 rows and returns the next 10 rows from the employees table.

SELECT \* FROM employees LIMIT 10 OFFSET 20;

-- Give all the distinct Brands who sales Mobile

SELECT DISTINCT Brand FROM flipkart\_mobiles;

-- Give all the distinct Brands and Color from Mobile database

SELECT DISTINCT Brand , Color FROM flipkart\_mobiles;

-- Give all the Unique Director who produces movie in Netflix.

SELECT DISTINCT director FROM netflix;

-- Give all the unique director name who produces movie in Nexflix and belongs to the country India

SELECT DISTINCT director , country FROM netflix WHERE country = "India";

## **Resources - Official Documentation and Other Resources**

* SQL Commands :- <https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/>
* ALTER TABLE - <https://www.w3schools.com/sql/sql_alter.asp>
* DROP TABLE - <https://www.w3schools.com/sql/sql_drop_table.asp>
* TURNCATE TABLE - <https://www.w3schools.com/sql/sql_ref_drop_table.asp>
* DROP DATABASES - <https://www.w3schools.com/sql/sql_drop_db.asp>
* Arithmetic Operators - <https://www.geeksforgeeks.org/sql-arithmetic-operators/>
* Compariosn Operators - <https://www.javatpoint.com/sql-comparison-operators>
* Logical Operators - <https://www.geeksforgeeks.org/sql-logical-operators/>
* SQL UNION Operator - <https://www.w3schools.com/sql/sql_union.asp>
* SQL UNION ALL Operator - <https://www.w3schools.com/sql/sql_ref_union_all.asp>
* LIMIT - <https://www.w3schools.com/mysql/mysql_limit.asp>
* ORDER BY - <https://www.w3schools.com/sql/sql_orderby.asp>
* DISTINCT - <https://www.w3schools.com/sql/sql_distinct.asp>