ALL THE SQL QUERRIES AND THEIR SYNTAX

There are five types of SQL commands:

DDL, DML, DCL, TCL, DQL.

- ## Data Defination Language:

1. **CREATE:** It is used to create a new table in the database.

Syntax:

CREATE TABLE TABLE_NAME (COLUMN_NAME1 DATATYPES(size)s , COLUMN_NAME2 DATATYPES(size)s ---------- COLUMN_NAMEN DATATYPES(size)s);

Example: CREATE TABLE EMP(id int(220), name varchar (200),..);

2. **DROP** : This statement is used to drop an existing database or Table. When you use this statement, complete information present in the database/ Table will be lost.
   Syntax :
   DROP DATABASE DatabaseName;
   DROP TABLE TableName;

3. ALTER :

This command is used to delete, modify or add constraints or columns in an existing table.

The **'ALTER TABLE'** Statement: This statement is used to **add, delete, modify columns** in an existing table.

The **'ALTER TABLE'** Statement : with **ADD/DROP COLUMN** You can use the ALTER TABLE statement with ADD/DROP Column. If you wish to add a column, then you will use the **ADD** command, and if you wish to delete a column, then you will use the **DROP** COLUMN command.

Syntax

- ALTER TABLE TableName **ADD** ColumnName Datatype;
- **ALTER** TABLE TableName **DROP** COLUMN ColumnName;

The 'ALTER TABLE' Statement: with **ALTER/MODIFY COLUMN** This statement is used to change the datatype of an existing column in a table.

Syntax :

ALTER TABLE TableName ADD COLUMN ColumnName Datatype;

4. **TRUNCATE** This command is used to delete the information present in the table but does not delete the table. So, once you use this command, your information will be lost, but not the table.

Syntax: TRUNCATE TABLE table_name;

• Data Manipulation Language

Insert

Update

Delete

1. **INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

INSERT INTO TABLE_NAME (col1s col2s col3s.... col N) VALUES (value1s value2s value3s .... valueN); Or INSERT INTO TABLE_NAME VALUES (value1s value2s value3s .... valueN);

2. **UPDATE:** This command is used to update or modify the value of a column in the table.

Syntax:

UPDATE table_name SET column1= values column2= values columnN = value WHERE CONDITION;

3. **DELETE:** It is used to remove one or more row from a table. **Syntax:**

DELETE FROM table_name;

DELETE FROM table_name WHERE condition;

1) Data Control Language:

DCL commands are used to grant and tase bacs authority from any database user. Here are some commands that come under DCL:

1. **Grant :** It is used to give user access privileges to a database.

GRANT SELECTs UPDATE ON MY_TABLE TO SOME_USERs ANOTHER_USER;

2. Revoke:-

It is used to tase bacs permissions from the user.

REVOKE SELECT UPDATE ON MY_TABLE FROM USER1s USER2.

2

2) Transaction Control Language :

TCL commands can only use with DML commands like **INSERT, DELETE and UPDATE only.** These operations are automatically committed in the database that's why they **cannot be used while creating tables or dropping them.**

Here are some commands that come under TCL:

o COMMIT o ROLLBACK;

1. **Commit:** Commit command is used to save all the transactions to the database.

Syntax: **COMMIT;**

Example: **DELETE** FROM CUSTOMERS WHERE AGE = 25;

COMMIT;

2. **Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

Syntax: **ROLLBACK;**

DELETE FROM CUSTOMERS WHERE AGE = 25;

ROLLBACK;

3) Data Query Language:

DQL is used to fetch the data from the database.

EX. **SELECT**

SELECT Column1, Column2, ...ColumN FROM TableName; --

1) (*) is used to select all from the table SELECT * FROM table_name

To select the number of records to return use:

SELECT TOP 3 * FROM Table

Other querries

DISTINCT

ORDER BY

GROUP BY

HAVING Clause

INTO

Command	Description	Syntax	Example
Select	The SELECT statement is the primary command used to retrieve data from a database	SELECT column1, column2 FROM table_name;	SELECT first_name, last_name FROM customers;
Where	The WHERE clause is used to filter rows based on a specified condition.	SELECT * FROM table_name WHERE condition;	SELECT * FROM customers WHERE age > 30; SELECT * FROM Products WHERE Price > 30; SELECT * FROM Products WHERE Price >= 30;
Order By	The ORDER BY clause is used to sort the result set in ascending or descending order based on a specified column.	SELECT * FROM table_name ORDER BY column_name ASC|DESC;	SELECT * FROM products ORDER BY price DESC;

Group BY	The GROUP BY clause groups rows based on the values in a specified column. It is often used with aggregate functions like **COUNT, SUM, AVG, etc.**	SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;	SELECT category, COUNT(*) FROM products GROUP BY category;

1. The **'SELECT DISTINCT'** Statement This statement is used to return only different values.

Syntax :

SELECT **DISTINCT** Column1, Column2, ...ColumnN FROM TableName;

2. The **'ORDER BY'** Statement The 'ORDER BY' statement is used to sort the required results in ascending or descending order. The results are sorted in ascending order by default.

SELECT Column1, Column2, ...Column n FROM TableName **ORDER BY** Column1, Column2, ... ASC|DESC;

3. The **'GROUP BY'** Statement. This 'GROUP BY' statement is used with the aggregate functions to group the result-set by one or more columns.

Syntax :

SELECT Column1, Column2,..., ColumnN FROM TableName WHERE Condition **GROUP BY** ColumnName(s) ORDER BY ColumnName(s);

4. The 'HAVING' Clause. The 'HAVING' clause is used in SQL because the WHERE keyword cannot be used everywhere.

   Syntax:

SELECT ColumnName(s) FROM TableName WHERE Condition GROUP BY ColumnName(s) **HAVING** Condition ORDER BY ColumnName(s);

5. The **'SELECT INTO'** Statement: The 'SELECT INTO' statement is used to copy data from one table to another.

Syntax

SELECT * INTO NewTable IN ExternalDB FROM OldTable WHERE Condition;

querry	Purpose	syntax	Example
IN	The IN command is used to determine whether a value matches any value in a subquery result. It is often used in the WHERE clause.	SELECT column(s) FROM table WHERE value IN (subquery);	SELECT * FROM Customers WHERE City IN ('Paris','London');
ANY	The ANY command is used to compare a value to any value returned by a subquery. It can be used with comparison operators like =, >, < ANY (subquery);	SELECT column(s) FROM table WHERE value < ANY (subquery);	SELECT * FROM Products WHERE Price > ANY (SELECT Price FROM Products WHERE Price > 50);
ALL	The ALL command is used to compare a value to all values returned by a subquery. It can be used with comparison operators like =, >, ALL (subquery);	SELECT column(s) FROM table WHERE value > ALL (subquery);	SELECT ProductName FROM Products WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

4) Subqueries in SQL Command Description :

1. AND QUERRY

```
SELECT column_name(s)
FROM table_name
WHERE column_1 = value_1
  AND column_2 = value_2;
```

AND is an operator that combines two conditions. Both conditions must be true for the row to be included in the result set.

2. AS

```
SELECT column_name AS 'Alias'
FROM table_name;
```

AS is a keyword in SQL that allows you to rename a column or table using an *alias*

3. AVG()

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

AVG() is an aggregate function that returns the average value for a numeric column.

4. BETWEEN

```
SELECT column_name(s)
FROM table_name
```

WHERE column_name BETWEEN value_1 AND value_2;

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

EX. SELECT * FROM Products
WHERE Price BETWEEN 50 AND 60;

5. IS NULL / IS NOT NULL

SELECT column_name(s)
FROM table_name
WHERE column_name IS NULL;

IS NULL and IS NOT NULL are operators used with the WHERE clause to test for empty values.

6. LIKE

SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;

LIKE is a special operator used with the WHERE clause to search for a specific pattern in a column.

EX. SELECT * FROM Customers
WHERE City LIKE 's%';

i. NOT

EX. SELECT * FROM Customers

WHERE City NOT LIKE 's%';

7. LIMIT

```
SELECT column_name(s)
FROM table_name
LIMIT number;
```

LIMIT is a clause that lets you specify the maximum number of rows the result set will have.

8. MAX()

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

MAX() is a function that takes the name of a column as an argument and returns the largest value in that column.

9. MIN()

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

MIN() is a function that takes the name of a column as an argument and returns the smallest value in that column.

10. OR

```
SELECT column_name
FROM table_name
WHERE column_name = value_1
  OR column_name = value_2;
```

OR is an operator that filters the result set to only include rows where either condition is true.

EX. SELECT * FROM Customers
WHERE City = "London" OR Country = "UK";

11. SUM

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

SUM() is a function that takes the name of a column as an argument and returns the sum of all the values in that column.

12.UPDATE

```
UPDATE table_name
SET some_column = some_value
WHERE some_column = some_value;
```

UPDATE statements allow you to edit rows in a table.

SQl Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Ex.

=

```
SELECT * FROM Products WHERE sale_date='2008-11-11'
```

TASK TO PERFORM QUERRIES

- CREATE A DATABSE WHERE TABLE IS PRESENT WHOSE ENTITY IS" SALES " AND IT IS HAVING MAXIMUM 5-6 RECORDS WHICH IS HAVING ATTRIBUTES LIKE SALE_ID, PRODUCT_ID, PROD_NAME, QUANTITY SOLD, TOTAL PRICE, LOCATION.

//create First table Sales

```
= CREATE TABLE Sales (
   sale_id INT PRIMARY KEY,
   product_id INT,
   quantity_sold INT,
   sale_date DATE,
   total_price DECIMAL(10, 2)
   FOREIGN KEY (product_id) REFERENCES Products(product_id)
);

-- Insert sample data into Sales table
```

```
INSERT INTO Sales (sale_id, product_id, quantity_sold,
sale_date, total_price) VALUES
(1, 101, 5, '2024-01-01', 2500.00),
(2, 102, 3, '2024-01-02', 900.00),
(3, 103, 2, '2024-01-02', 60.00),
(4, 104, 4, '2024-01-03', 80.00),
(5, 105, 6, '2024-01-03', 90.00);
```

// create Second table Product

```
CREATE TABLE Products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    unit_price DECIMAL(10, 2)
);

-- Insert sample data into Products table

INSERT INTO Products (product_id, product_name, category,
unit_price) VALUES
(101, 'Laptop', 'Electronics', 500.00),
(102, 'Smartphone', 'Electronics', 300.00),
(103, 'Headphones', 'Electronics', 30.00),
(104, 'Keyboard', 'Electronics', 20.00),
(105, 'Mouse', 'Electronics', 15.00);
```

Create a querries using above tables

- **Retrieve the product_name and unit_price from the Products table.**
  -SELECT product_name, unit_price FROM Products;

- **Filter the Sales table to show only sales with a total_price greater than------**
  -SELECT * FROM Sales WHERE total_price >-----;

- **Filter the SALES table to show only products_NAME AND USE WHERE AND CONDITION...**
- `SELECT * FROM Products WHERE category = 'Electronics';`
-

- **Retrieve the sale_id and total_price from the Sales table for sales made IN SPECIFIC LOCATION... EX. WHERE LOCATION IS MUMBAI/-----**

  EX. SELECT sale_id, total_price
  FROM Sales
  WHERE LOCATION = '------';

- **Retrieve the product_id and product_name from the SALES table with a TOTAL-price greater than------**
- EX. SELECT pROD_id, PROD-NAME
  FROM Sales
  WHERE TOTALPRICE >----';
-

- **Calculate the total revenue generated from all sales in the Sales table.**
- **Query:**

```
SELECT SUM(total_price) AS total_revenue
FROM Sales;
```

- **Calculate the average unit_price of products in the Products table.**

Query:

SELECT AVG(unit_price) AS average_unit_price
FROM Products;

- **. Calculate the total quantity_sold from the Sales table.**
- **Query:**
- SELECT SUM(quantity_sold) AS total_quantity_sold
FROM Sales;

- **Count Sales Per Day from the Sales table**
- **Query:**

```
SELECT sale_date, COUNT(*) AS sales_count
FROM Sales
GROUP BY sale_date
ORDER BY sale_date;
```

- **Retrieve product_name and unit_price from the Products table with the Highest Unit Price**

- **Query**:
- SELECT product_name, unit_price
  FROM Products
  ORDER BY unit_price DESC
  LIMIT 1;

- **Retrieve the sale_id, product_id, and total_price from the Sales table for sales with a quantity_sold greater than 4.**
  Query:
  SELECT sale_id, product_id, total_price
  FROM Sales
  WHERE quantity_sold > 4;

- **Retrieve the product_name and unit_price from the Products table, ordering the results by unit_price in descending order.**
- **Query:**
- SELECT product_name, unit_price
  FROM Products
  ORDER BY unit_price DESC;

- **Retrieve the total_price of all sales, rounding the values**

- **to two decimal places.**
- **Query:**

- SELECT ROUND(SUM(total_price), AS total_sales
  FROM Sales;
- **Calculate the average total_price of sales in the Sales table.**
- **Query:**
- ```
  SELECT AVG(total_price) AS average_total_price
  FROM Sales;
  ```

- **Retrieve the sale_id and sale_date from the Sales table, formatting the sale_date as 'YYYY-MM-DD'.**
- **Query:**
- ```
  SELECT sale_id, DATE_FORMAT(sale_date, '%Y-%m-%d') AS formatted_date
  FROM Sales;
  ```

- **Retrieve the product_name and category from the Products table, ordering the results by category in ascending order.**

  Query:

  SELECT product_name, category

  FROM Products

  ORDER BY category ASC;

- **Calculate the total quantity_sold of products in the 'Electronics' category.**

Query:

```
SELECT SUM(quantity_sold) AS
total_quantity_sold
FROM Sales
JOIN Products ON Sales.product_id =
Products.product_id
WHERE Products.category = 'Electronics';
```

- **Retrieve the product_name and total_price from the Sales table, calculating the total_price as quantity_sold multiplied by unit_price.**

  Query:

```
SELECT product_name, quantity_sold *
unit_price AS total_price
FROM Sales
JOIN Products ON Sales.product_id =
Products.product_id;
```

- **Find the Products Not Sold from Products table**

Query:

```
SELECT product_id, product_name
FROM Products
WHERE product_id NOT IN (SELECT DISTINCT
product_id FROM Sales);
```

- **Calculate the total revenue generated from sales for each product category.**

  Query:

```
SELECT p.category, SUM(s.total_price) AS
total_revenue
FROM Sales s
JOIN Products p ON s.product_id =
p.product_id
GROUP BY p.category;
```

- **Find the product category with the highest average unit price.**

  Query:

```
SELECT category
FROM Products
GROUP BY category
```

```
ORDER BY AVG(unit_price) DESC
LIMIT 1;
```

- **Identify products with total sales exceeding 30.**

  Query:

```
SELECT p.product_name
FROM Sales s
JOIN Products p ON s.product_id =
p.product_id
GROUP BY p.product_name
HAVING SUM(s.total_price) > 30;
```

- **Count the number of sales made in each month.**

  Query:

```
SELECT DATE_FORMAT(s.sale_date, '%Y-%m')
AS month, COUNT(*) AS sales_count
FROM Sales s
GROUP BY month;
```

- **Retrieve Sales Details for Products with 'Smart' in Their Name**

  Query:

  ```
  SELECT s.sale_id, p.product_name, s.total_price
  FROM Sales s
  JOIN Products p ON s.product_id =
  p.product_id
  WHERE p.product_name LIKE '%Smart%';
  ```

- **Determine the average quantity sold for products with a unit price greater than $100.**

  Query:

  ```
  SELECT AVG(s.quantity_sold) AS
  average_quantity_sold
  FROM Sales s
  JOIN Products p ON s.product_id =
  p.product_id
  WHERE p.unit_price > 100;
  ```

- **Retrieve the product name and total sales revenue for each product.**

  Query:

```
SELECT p.product_name, SUM(s.total_price) AS
total_revenue
FROM Sales s
JOIN Products p ON s.product_id =
p.product_id
GROUP BY p.product_name;
```

- **List all sales along with the corresponding product names.**

  Query:

```
SELECT s.sale_id, p.product_name
FROM Sales s
JOIN Products p ON s.product_id =
p.product_id;
```

- **Identify sales where the quantity sold is greater than the average quantity sold.**

  Query:

```
SELECT *
FROM Sales
```

```
WHERE quantity_sold > (SELECT
AVG(quantity_sold) FROM Sales);
```

- **Extract the month and year from the sale date and count the number of sales for each month.**

  Query:

```
SELECT CONCAT(YEAR(sale_date), '-',
LPAD(MONTH(sale_date), 2, '0')) AS month,
      COUNT(*) AS sales_count
FROM Sales
GROUP BY YEAR(sale_date),
MONTH(sale_date);
```

- **Calculate the number of days between the current date and the sale date for each sale.**

  Query:

```
SELECT sale_id, DATEDIFF(NOW(), sale_date) AS
days_since_sale
FROM Sales;
```

- **Identify sales made during weekdays versus weekends.**

  Query:

```
SELECT sale_id,
      CASE
         WHEN DAYOFWEEK(sale_date) IN (1, 7)
THEN 'Weekend'
         ELSE 'Weekday'
      END AS day_type
FROM Sales;
```