CLAUSES

1) Select	**2)From**	**3)Where**	**4)Join**
5) Group By	**6)Having**	**7)Order By**	

1) **Select:-**The SELECT clause is used to specify the columns to retrieve from a table.
Syntax:-

```
        SELECT first_name, last_name FROM employees;
```

2)**FROM:-**The FROM clause specifies the table or tables from which to retrieve data.
Syntax:-

```
                SELECT * FROM orders;
```

3)WHERE:-The **WHERE** clause is used to filter the rows returned by a query based on specified conditions.
Syntax:-

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

Example:-

```
              SELECT * FROM customers WHERE country = 'USA';
```

4) **JOIN:-**The **JOIN** clause combines rows from two or more tables based on a related column between them.
Syntax:-

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

GROUP BY:-The GROUP BY clause groups the result set based on one or more columns, typically used with aggregate functions.
Syntax:-

```
SELECT column1, SUM(column2)
FROM table_name
GROUP BY column1;
```

Example:-

```
       select id,sum(salary)
   -> from details
   -> group by id;
```

```
SELECT country, COUNT(*) as total_customers
FROM customers
GROUP BY country;
```

Assuming we have the following data in the "**Employees**" table:

EmployeeID	Department	Salary	Age
1	HR	50000	30
2	IT	60000	35
3	IT	55000	28
4	HR	45000	32
5	IT	70000	40

We can use the GROUP BY clause to calculate the average salary and maximum age for each department:

```
SELECT Department, AVG(Salary) AS AverageSalary, MAX(Age)
AS MaxAge
FROM Employees
GROUP BY Department;
```

```
Department | AverageSalary | MaxAge
-----------------------------------
   HR    |    47500     |  32
   IT    |   61666.67  |  40
```

5) **<u>HAVING</u>**<u>:-</u>The HAVING clause filters the groups created by the GROUP BY clause based on specified conditions.
Syntax:-

```
SELECT column1, SUM(column2)
FROM table_name
GROUP BY column1
HAVING condition;
```

Example:-

```
SELECT country, COUNT(*) as total_customers
FROM customers
GROUP BY country
HAVING COUNT(*) > 10;
```

6) **<u>Order By:-</u>**The ORDER BY clause is used to sort the result set based on one or more columns.
7)
Syntax:-

```
SELECT column1, column2
FROM table_name
ORDER BY column1 ASC;

SELECT column1, column2
FROM table_name
ORDER BY column1 desc;
```

Example:-

```
SELECT product_name, unit_price
FROM products
ORDER BY unit_price DESC;
SELECT *
```

FROM my_view;

8) Limit:-The **LIMIT** clause is used to restrict the number of rows returned by a query. It is commonly used with the **ORDER BY** clause to fetch **a specific number of rows** from the beginning.

Syntax:-
SELECT column1, column2
FROM table_name
LIMIT 10;

These are just a few examples of the clauses available in SQL. There are other clauses as well, such as **JOIN, UNION, INSERT INTO, UPDATE, DELETE,** and more, which serve different purposes in manipulating and managing data in a database.

VIEWS

To create a view, you typically use the **CREATE VIEW** statement. The exact syntax may vary depending on the database system you are using.
Here's a syntax of creating a simple view:

```
CREATE VIEW my_view AS
SELECT column1, column2, ...
FROM table1
WHERE condition;
```

Example:-

Once the view is created, you can query it like a regular table using the **SELECT** statement:

```
SELECT *
FROM my_view;
```

Views are a powerful feature of SQL that provide flexibility, security, and abstraction when working with complex data structures.
They help simplify queries, enhance data security, and promote code reuse by encapsulating frequently used logic into a single view.

```
mysql> create view my_views as
    -> select id,salary
    -> from details
    -> where rollnumber=570;

    mysql> select *
       -> from my_views;
```

STORED PROCEDURES

SYNTAX:-

```
CREATE PROCEDURE procedure_name
    [parameter_list]
    [RETURNS data_type]
BEGIN
    -- Procedure body (SQL statements and procedural logic)
END;
```

Let's create a stored procedure that calculates the average salary of employees within a specified department in the "details" table.

Here's an example of a stored procedure that takes the department name as input and calculates the average salary for that department:

Let's break down each part:

Let's break down each part:

```
DELIMITER //

MySQL> CREATE TABLE employees (
    ->     id INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     age INT,
    ->     department VARCHAR(50),
    ->     salary DECIMAL(10,2)
    -> );
```

```
mysql> DELIMITER //
mysql>  CREATE PROCEDURE getdetailsofemployees(in gen varchar(50))
    -> begin
    -> select * from details where gender=gen;
    -> end//
Query OK, 0 rows affected (0.01 sec)

mysql> call getdetailsofemployees('male');
    -> //
```

In this example, the stored procedure "CalculateAverageSalaryByDepartmentName" is created with two parameters: "department_name" as the input parameter and "average_salary" as the output parameter. The procedure calculates the average salary of employees in the specified department and assigns the result to the output parameter.

To execute the stored procedure and retrieve the average salary for a specific department, you can use the CALL statement:

```
SET @avg_salary = 0;
CALL CalculateAverageSalaryByDepartmentName('IT', @avg_salary);
SELECT @avg_salary AS AverageSalary;
```