SELECT :

CLASS 1 :

use employees;

- - Select * FROM employees;
- - SELECT first_name, last_name, EMP_NO FROM employees;
- - SELECT first_name, last_name FROM employees.employees;

CLASS 2:

REPLACE OR WITH AND :

use employees;

- - SELECT * FROM EMPLOYEES WHERE GENDER = 'M';
  -- SELECT first_name, last_name FROM employees WHERE BIRTH_DATE > 15;

```
SELECT first_name
FROM employees
WHERE hire_date > '2015-01-01' OR  gender = 'F';
```

- - SELECT emp_no, first_name, last_name
  -- FROM employees
  -- WHERE birth_date < '1990-01-01' OR first_name LIKE 'J%';

CLASS 3 :

- - SELECT * FROM employees;
- - SELECT first_name
  -- FROM employees

-- WHERE first_name LIKE 'D%';

- - SELECT last_name
  -- FROM employees
  -- WHERE last_name LIKE '%son%';
- - SELECT first_name
  -- FROM employees
  -- WHERE first_name LIKE '___a%';
- - SELECT first_name
  -- FROM employees
  -- WHERE first_name LIKE 'J_N%';

```
SELECT first_name
FROM employees
WHERE first_name LIKE '%an%';
```

class 4 :

```
use employees;

SELECT first_name, last_name, gender
FROM employees
WHERE gender IN ('M', 'F');

SELECT first_name, last_name, hire_date
FROM employees
WHERE YEAR(hire_date) NOT IN (2010, 2015, 2020);

SELECT first_name, last_name, gender
FROM employees
WHERE gender NOT IN ('M', 'F');
```

- - null or not null

```
SELECT first_name, last_name, birth_date
FROM employees
WHERE birth_date IS NULL;
```

```
SELECT first_name, last_name, gender
FROM employees
WHERE gender IS NOT NULL;
```

- - between

```
SELECT first_name, last_name, birth_date
FROM employees
WHERE birth_date BETWEEN '1980-01-01' AND '1990-12-31';
```

```
SELECT first_name, last_name, emp_no
FROM employees
WHERE emp_no BETWEEN 1000 AND 2000;
```

```
SELECT first_name, last_name, birth_date, hire_date
FROM employees
WHERE birth_date BETWEEN '1980-01-01' AND '1990-12-31'
AND hire_date > '2000-01-01';
```

```
SELECT first_name, last_name, gender, emp_no, hire_date
FROM employees
WHERE (gender = 'M' OR hire_date < '1990-01-01')
AND emp_no BETWEEN 1000 AND 3000;
```

class 5 :

- - SELECT * FROM employees;
- - SELECT COUNT(*) FROM employees;
- - SELECT SUM(emp_no) FROM employees;
- - SELECT AVG(emp_no) FROM employees;
- - alias :
- - SELECT first_name AS "f", last_name AS "Last Name" FROM employees;

```
SELECT e.first_name, e.last_name
FROM employees AS e
WHERE e.gender = 'M';
```

class 6 :

- - SELECT first_name, hire_date
  -- FROM employees
  -- ORDER BY hire_date ASC ;
- - SELECT first_name, last_name, gender
  -- FROM employees
  -- ORDER BY gender DESC, last_name ASC;
- - order by
- - SELECT gender, COUNT(*) AS employee_count
  -- FROM employees
  -- GROUP BY gender;

```
SELECT gender, AVG(DATEDIFF(CURDATE(), birth_date)/365) AS avg_age
FROM employees
GROUP BY gender;employees
```

class 7 : advance filtering techniques :
1. Having

2. Lmit
3. distinct

- - SELECT gender, COUNT*() AS employee_count*
  -- FROM employees
  -- GROUP BY gender
  -- HAVING COUNT(
) > 9;
- - SELECT first_name, MAX(birth_date) AS latest_birthday
  -- FROM employees
  -- GROUP BY first_name
  -- HAVING MAX(birth_date) < '1990-01-01';
- - SELECT * FROM employees
  -- LIMIT 5;

SELECT * FROM employees
LIMIT 5 OFFSET 5;

SELECT DISTINCT gender FROM employees;

SELECT DISTINCT emp_no , gender FROM employees;

SELECT DISTINCT first_name, last_name
FROM employees;

Yes, the query you provided will work if you want to fetch the `first_name` and `last_name` from the `employees` table inside the `employees` database. Here's your query for reference:

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

---

1. Without Specifying the Database

If you've already executed `USE employees;`, you don't need to specify the database name repeatedly. You can simplify it to:

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

2. Fully Qualified Query

If you don’t use `USE employees;` and want to explicitly reference the database and table, use:

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

3.

If you run the following command:

```
SELECT * FROM employees;
```

1. SELECT with WHERE

The `WHERE` clause is used to filter rows based on specified conditions.

- **Example 1**: Filter employees in the "Sales" department.

```
SELECT * FROM employees WHERE GENDER = 'M';
```

  Explanation: Retrieves all employees working in the "Sales" department.

- **Example 2**: Employees older than 30.

```
SELECT first_name, last_name FROM employees WHERE BIRTH_DA
TE > 15;
```

  Explanation: Retrieves the first name and last name of employees whose age is greater than 30.

1. Using AND (Multiple Conditions: All conditions must be true)

Example 1: Filter by `hire_date` after a specific date and `gender` is male

```
sql
Copy code
SELECT first_name, last_name
FROM employees
WHERE hire_date > '2015-01-01' AND gender = 'M';
```

Explanation: This query retrieves the first and last names of male employees who were hired after January 1st, 2015.

Example 2: Filter by `birth_date` and `first_name`

```
sql
Copy code
SELECT emp_no, first_name, last_name
FROM employees
WHERE birth_date < '1990-01-01' AND first_name LIKE 'J%';
```

Explanation: This query retrieves the employee number, first name, and last name of employees born before January 1st, 1990, and whose first name starts with "J".

---

2. Using OR (Multiple Conditions: At least one condition must be true)

Example 1: Filter by `gender` or `hire_date`

```
sql
Copy code
SELECT first_name, last_name
FROM employees
WHERE gender = 'F' OR hire_date > '2020-01-01';
```

Explanation: This query retrieves the first and last names of female employees or employees hired after January 1st, 2020.

Example 2: Filter by `birth_date` or `last_name`

```
sql
Copy code
SELECT emp_no, birth_date, last_name
FROM employees
WHERE birth_date > '1995-01-01' OR last_name LIKE 'S%';
```

Explanation: This query retrieves the employee number, birth date, and last name of employees who were born after January 1st, 1995, or whose last name starts with "S".

1. `LIKE` with `%` (Any Sequence of Characters)

Example 1: Find employees whose `first_name` starts with "A"

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'A%';
```

Explanation: Retrieves employees whose first name starts with the letter "A".

Example 2: Find employees whose `last_name` contains "son"

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%son%';
```

Explanation: Retrieves employees whose last name contains the substring "son".

---

2. `LIKE` with `_` (Single Character)

Example 1: Find employees whose `first_name` has "a" as the second character

```
SELECT first_name
```

```
FROM employees
WHERE first_name LIKE '_a%';
```

Explanation: Retrieves employees whose first name has "a" as the second character (e.g., "Sarah", "Daniel").

Example 2: Find employees whose `last_name` has "a" as the third character

```
SELECT last_name
FROM employees
WHERE last_name LIKE '__a%';
```

Explanation: Retrieves employees whose last name has "a" as the third character (e.g., "Baker", "Gavin").

---

3. `LIKE` with `%` and `_` Combination (Pattern Matching with Specific Character Positions)

Example 1: Find employees whose `first_name` starts with "J" and the second character is "o"

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'J_o%';
```

Explanation: Retrieves employees whose first name starts with "J" and has "o" as the second character (e.g., "John", "Jon").

Example 2: Find employees whose `last_name` starts with "D" and has "n" as the second character

```
SELECT last_name
FROM employees
WHERE last_name LIKE 'D_n%';
```

Explanation: Retrieves employees whose last name starts with "D" and has "n" as the second character (e.g., "Dean", "Dunn").

---

4. `LIKE` with `%` for Partial Matching

Example 1: Find employees whose `first_name` contains "an"

```
SELECT first_name
FROM employees
WHERE first_name LIKE '%an%';
```

Explanation: Retrieves employees whose first name contains the substring "an" (e.g., "Daniel", "Samantha").

Example 2: Find employees whose `last_name` contains "er"

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%er%';
```

Explanation: Retrieves employees whose last name contains the substring "er" (e.g., "Baker", "Miller").

5. `SELECT` with `ORDER BY`

The `ORDER BY` keyword is used to sort the result set in ascending (`ASC`) or descending (`DESC`) order.

Example 1: Sort by `hire_date` in ascending order

```
SELECT first_name, hire_date
FROM employees
ORDER BY hire_date ASC;
```

Explanation: Retrieves the first name and hire date of employees, sorted by the `hire_date` in ascending order (earliest to latest).

Example 2: Sort by `gender` in descending order, then by `last_name` in ascending order

```
SELECT first_name, last_name, gender
FROM employees
ORDER BY gender DESC, last_name ASC;
```

Explanation: Retrieves employees' first name, last name, and gender, sorted by gender in descending order (with "F" first), and then by `last_name` in ascending order in case of ties.

---

6. `SELECT` with `GROUP BY`

The `GROUP BY` clause is used to group rows that have the same values in specified columns.

Example 1: Count employees by `gender`

```
sql
Copy code
SELECT gender, COUNT(*) AS employee_count
FROM employees
```

```
GROUP BY gender;
```

Explanation: Counts the number of employees in each gender group.

Example 2: Average `age` by `gender`

```
SELECT gender, AVG(DATEDIFF(CURDATE(), birth_date)/365) AS av
g_age
FROM employees
GROUP BY gender;
```

Explanation: Calculates the average age of employees by gender. (Assuming you use `birth_date` to calculate age using the `DATEDIFF` function).

1. SELECT with HAVING

- **Purpose**: Filters results after using `GROUP BY` with aggregate functions (e.g., `COUNT()`, `AVG()`).
- **When to use**: Use when you need to apply conditions on aggregated data.

Example 1: Departments with more than 10 employees.

```
SELECT department, COUNT(*) AS employee_count
FROM employees
GROUP BY department
HAVING COUNT(*) > 10;
```

Example 2: Departments where the average salary is above 50,000.

```
SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

2. SELECT with LIMIT

- **Purpose**: Limits the number of rows returned in the result set.
- **When to use**: Use when you want to restrict the number of rows returned (e.g., for pagination).

Example 1: Get only the first 5 employees.

```
SELECT * FROM employees
LIMIT 5;
```

Example 2: Get employees 6 through 10.

```
SELECT * FROM employees
LIMIT 5 OFFSET 5;
```

3. SELECT with DISTINCT

- **Purpose**: Removes duplicate rows from the result set, returning only unique values.
- **When to use**: Use when you want only distinct values in your result set.

Example 1: Get distinct department names.

```
SELECT DISTINCT department FROM employees;
```

Example 2: Get distinct combinations of city and department.

```
SELECT DISTINCT emp_no , gender FROM employees;
```

These clauses (`HAVING`, `LIMIT`, and `DISTINCT`) allow for more advanced data filtering and result management when working with SQL.

10. Aggregation Functions

5. COUNT(), SUM(), MIN(), MAX(), AVG()

These are common aggregate functions:

- `COUNT()`: Counts the number of rows.
- `SUM()`: Adds up values in a column.
- `MIN()`: Finds the smallest value.
- `MAX()`: Finds the largest value.
- `AVG()`: Finds the average value.

Aggregation functions are used to perform calculations on multiple rows of a single column and return a single output.

Here are examples for each of the aggregate functions: `COUNT()`, `SUM()`, `MIN()`, `MAX()`, and `AVG()`.

1. COUNT()

Counts the number of rows in a specified column or table.

Example:

```
sql
Copy code
SELECT COUNT(*) FROM employees;
```

Explanation: This query counts the total number of rows in the `employees` table.

2. SUM()

Adds up all values in a specified numeric column.

Example:

```
sql
Copy code
SELECT SUM(salary) FROM employees;
```

Explanation: This query calculates the total sum of all `salary` values in the `employees` table.

3. MIN()

Finds the smallest value in a specified column.

Example:

```
sql
Copy code
SELECT MIN(salary) FROM employees;
```

Explanation: This query returns the minimum salary from the `employees` table.

4. MAX()

Finds the largest value in a specified column.

Example:

```
sql
Copy code
SELECT MAX(salary) FROM employees;
```

Explanation: This query returns the maximum salary from the `employees` table.

5. AVG()

Calculates the average of a specified numeric column.

Example:

```
sql
Copy code
SELECT AVG(salary) FROM employees;
```

Alias in SQL

An **alias** is a temporary name that is used to rename a table or column for the duration of a query. This makes the result set easier to read and more concise.

- `AS` is the keyword used to assign an alias to a column or table.

Examples for Alias:

1. Alias for Columns

You can use aliases for columns to give them more readable names in the result set.

Example: Rename `first_name` to `First Name` in the result set

```
sql
Copy code
SELECT first_name AS "First Name", last_name AS "Last Name" F
ROM employees;
```

Explanation: This query selects `first_name` and `last_name` from the `employees` table and renames them to "First Name" and "Last Name" for the result set.

2. Alias for Tables

You can also assign aliases to tables to make your SQL queries shorter and easier to read, especially when working with joins.

Example: Use an alias for the `employees` table

```
sql
Copy code
SELECT e.first_name, e.last_name
FROM employees AS e
WHERE e.gender = 'M';
```

`SELECT` with `IN`

The `IN` operator allows you to specify multiple values in a `WHERE` clause.

Example 1: Find employees whose `gender` is either "M" or "F"

```
SELECT first_name, last_name, gender
FROM employees
WHERE gender IN ('M', 'F');
```

Explanation: Retrieves the first name, last name, and gender of employees whose gender is either "M" (Male) or "F" (Female).

Example 2: Find employees whose `hire_date` is in the years 2010, 2015, or 2020

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE YEAR(hire_date) IN (2010, 2015, 2020);
```

Explanation: Retrieves the first name, last name, and hire date of employees hired in the years 2010, 2015, or 2020.

---

8. `SELECT` with `NOT IN`

The `NOT IN` operator allows you to exclude specific values in a `WHERE` clause.

Example 1: Find employees whose `gender` is not "M" or "F"

```
SELECT first_name, last_name, gender
FROM employees
WHERE gender NOT IN ('M', 'F');
```

Explanation: Retrieves employees whose gender is neither "M" (Male) nor "F" (Female). This might be useful if there are any other gender values.

Example 2: Find employees who were not hired in 2010, 2015, or 2020

```
SELECT first_name, last_name, hire_date
FROM employees
```

```
WHERE YEAR(hire_date) NOT IN (2010, 2015, 2020);
```

Explanation: Retrieves the first name, last name, and hire date of employees who were not hired in 2010, 2015, or 2020.

`SELECT` with `IS NULL`

The `IS NULL` operator is used to check if a column has a `NULL` value.

Example 1: Find employees whose `birth_date` is `NULL`

```
SELECT first_name, last_name, birth_date
FROM employees
WHERE birth_date IS NULL;
```

Explanation: Retrieves the first name, last name, and birth date of employees who do not have a birth date (i.e., their `birth_date` is `NULL`).

Example 2: Find employees whose `hire_date` is `NULL`

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE hire_date IS NULL;
```

Explanation: Retrieves the first name, last name, and hire date of employees who do not have a hire date (i.e., their `hire_date` is `NULL`).

---

10. `SELECT` with `IS NOT NULL`

The `IS NOT NULL` operator is used to check if a column does **not** have a `NULL` value.

Example 1: Find employees whose `gender` is not `NULL`

```
SELECT first_name, last_name, gender
FROM employees
WHERE gender IS NOT NULL;
```

Explanation: Retrieves the first name, last name, and gender of employees who have a specified gender (i.e., their `gender` is not `NULL`).

Example 2: Find employees who have a `birth_date`

```
SELECT first_name, last_name, birth_date
FROM employees
WHERE birth_date IS NOT NULL;
```

Explanation: Retrieves the first name, last name, and birth date of employees who have a specified birth date (i.e., their `birth_date` is not `NULL`).

`SELECT` with `BETWEEN`

The `BETWEEN` operator is used to filter the result set within a certain range. It is inclusive, meaning the values at the ends of the range are also included.

Example 1: Find employees whose `birth_date` is between two specific dates

```
SELECT first_name, last_name, birth_date
FROM employees
WHERE birth_date BETWEEN '1980-01-01' AND '1990-12-31';
```

Explanation: Retrieves the first name, last name, and birth date of employees who were born between January 1, 1980, and December 31, 1990.

Example 2: Find employees whose `emp_no` is between 1000 and 2000

```
SELECT first_name, last_name, emp_no
FROM employees
WHERE emp_no BETWEEN 1000 AND 2000;
```

Explanation: Retrieves the first name, last name, and employee number (`emp_no`) of employees whose employee number falls between 1000 and 2000.

Combining `AND`, `OR`, and `BETWEEN`

You can combine `AND`, `OR`, and `BETWEEN` to create more complex queries.

Example 1: Find employees whose `birth_date` is between 1980 and 1990 and who were hired after 2000

```
SELECT first_name, last_name, birth_date, hire_date
FROM employees
WHERE birth_date BETWEEN '1980-01-01' AND '1990-12-31'
AND hire_date > '2000-01-01';
```

Explanation: Retrieves the first name, last name, birth date, and hire date of employees born between 1980 and 1990 and hired after January 1, 2000.

Example 2: Find employees whose `gender` is "M" or who were hired before 1990, and whose `emp_no` is between 1000 and 3000

```
SELECT first_name, last_name, gender, emp_no, hire_date
FROM employees
```

```
WHERE (gender = 'M' OR hire_date < '1990-01-01')
AND emp_no BETWEEN 1000 AND 3000;
```

Explanation: Retrieves the first name, last name, gender, employee number, and hire date of employees who are male or were hired before January 1, 1990, and whose employee number is between 1000 and 3000.