```
create database db2;

use db2;

CREATE TABLE employees ( employee_id INT PRIMARY KEY, first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL, hire_date DATE, salary DECIMAL(10,2) );

insert into employees (employee_id, first_name, last_name, hire_date, salary) Values (1, 'f1' , 'l1',
'2025-02-10', 60000), (2, 'f2' , 'l2', '2025-02-12', 65000);


SELECT * FROM employees;

SELECT first_name, last_name, salary FROM employees;

SELECT employee_id AS emp_id, first_name AS fname, salary AS monthly_salary FROM employees;

insert into employees (employee_id, first_name, last_name, hire_date, salary) Values (3, 'f1' , 'l3',
'2025-02-10', 60000),
 (4, 'f2' , 'l4', '2025-02-12', 65000);

SELECT first_name AS fname, salary AS monthly_salary FROM employees;

SELECT DISTINCT first_name AS fname, salary AS monthly_salary FROM employees;

SELECT first_name, last_name, salary FROM employees WHERE
salary > 60000;

SELECT * from employees where salary > 50000 and hire_date
 > '2025-02-11';

SELECT first_name, last_name FROM employees WHERE last_name IN ('L2', 'l2', 'll');

SELECT first_name, last_name FROM employees WHERE first_name LIKE '%1';


SELECT first_name, last_name, salary FROM employees WHERE
salary BETWEEN 40000 AND 60000;


SELECT first_name, last_name, salary FROM employees ORDER
BY salary DESC;

SELECT first_name, last_name, salary FROM employees ORDER
BY salary aSC;

SELECT first_name, last_name FROM employees ORDER BY salary DESC LIMIT 3;
```

```
SELECT salary, COUNT(*) AS num_of_employees
   -> FROM employees
   -> GROUP BY salary;
```

`COUNT()` – Counts rows
`SUM()` – Sums up numeric values
`AVG()` – Averages numeric values
`MIN()` – Finds the minimum value
`MAX()` – Finds the maximum value

```
SELECT salary, COUNT(*) AS num_of_employees FROM employees GROUP BY salary HAVING
COUNT(*) > 1;

drop table employees;

CREATE TABLE employees (     employee_id INT PRIMARY KEY,
   first_name  VARCHAR(50) NOT NULL,     last_name   VARCHAR(50) NOT NULL,     department_id
INT,       salary      DECIMAL(10,2));

drop table employees;

CREATE TABLE departments (     department_id INT PRIMARY KEY,     department_name
VARCHAR(100) NOT NULL );

CREATE TABLE employees (     employee_id INT PRIMARY KEY,
   first_name  VARCHAR(50) NOT NULL,     last_name   VARCHAR(50) NOT NULL,     department_id
INT,       salary      DECIMAL(10,2));

drop table employees;
CREATE TABLE employees (     employee_id INT PRIMARY KEY,
   first_name  VARCHAR(50) NOT NULL,     last_name   VARCHAR(50) NOT NULL,     department_id int,
foreign key (department_id) REFEReNCES departments(department_id),       salary      DECIMAL(10,2));


INSERT INTO departments (department_id, department_name)
VALUES
(1, 'Sales'),
(2, 'Marketing'),
(3, 'Finance'),
(4, 'IT');

INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)
VALUES
(101, 'John',  'Smith',   1, 60000.00),
(102, 'Jane',  'Doe',     1, 65000.00),
(103, 'Mark',  'Brown',   2, 45000.00),
(104, 'Lucy',  'Davis',   3, 52000.00),
```

```
(105, 'Lisa',  'Johnson', NULL, 48000.00),
(106, 'Tom',   'Wilson',  4, 72000.00);
```

INNER JOIN only returns rows that have matching values in both tables.

```
SELECT e.employee_id, e.first_name, e.last_name,
      d.department_id, d.department_name
FROM employees AS e
INNER JOIN departments AS d
   ON e.department_id = d.department_id;
```

A **LEFT JOIN** (or **LEFT OUTER JOIN**) returns all rows from the left table, even if there's **no matching row** in the right table. If no match is found, columns from the right table are returned as `NULL`.

```
SELECT e.employee_id, e.first_name, e.last_name,
      d.department_id, d.department_name
FROM employees AS e
LEFT JOIN departments AS d
   ON e.department_id = d.department_id;
```

A **RIGHT JOIN** (or **RIGHT OUTER JOIN**) is the mirror opposite of **LEFT JOIN**. It returns all rows from the right table, plus matched rows from the left table.

```
SELECT d.department_id, d.department_name,
      e.employee_id, e.first_name, e.last_name
FROM employees AS e
RIGHT JOIN departments AS d
   ON e.department_id = d.department_id;
```

A **CROSS JOIN** returns the **Cartesian product** of the two tables—i.e., every row in the left table is paired with every row in the right table.

```
SELECT e.employee_id, e.first_name, d.department_id, d.department_name

FROM employees AS e

CROSS JOIN departments AS d;
```

FULL OUTER JOIN

MySQL **does not** have a native `FULL OUTER JOIN` keyword. However, you can mimic a **full join** using a combination of **LEFT JOIN** and **RIGHT JOIN** with `UNION`:

```
SELECT e.employee_id, e.first_name, e.last_name,
      d.department_id, d.department_name
FROM employees AS e
```

```
LEFT JOIN departments AS d
   ON e.department_id = d.department_id

UNION

SELECT e.employee_id, e.first_name, e.last_name,
      d.department_id, d.department_name
FROM employees AS e
RIGHT JOIN departments AS d
   ON e.department_id = d.department_id;
```