3.1 Perform DML Operations

a) Insert new employee

Let’s insert a new record into the Employee table.

```
INSERT INTO Employee (EmpID, EmpName, Dept, Salary, JoiningDate, City)
VALUES (111, 'Laura King', 'Finance', 85000.00, '2024-04-10', 'Boston');
```

✅ Explanation:

A new employee “Laura King” has been added to the Finance department.

b) Update salary of employees in IT department by 10%

We’ll increase the salary of all IT employees by 10%.

```
UPDATE Employee
SET Salary = Salary * 1.10
WHERE Dept = 'IT';
```

✅ Explanation:

All employees in the IT department (EmpID: 101, 104, 106, 110) will get a 10% salary increase.

c) Delete employees who joined before 2015

We’ll remove employees who joined before January 1, 2015.

```
DELETE FROM Employee
WHERE JoiningDate < '2015-01-01';
```

✅ Explanation:

This query deletes employees who joined before 2015, e.g., Jack Turner (EmpID 110).

✅ Result Summary After Operations

EmpID	EmpName	Dept	Salary	JoiningDate	City
101	Alice Johnson	IT	77000.00	2023-05-01	New York
102	Bob Smith	HR	55000.00	2018-03-15	Chicago
103	Carol White	Finance	80000.00	2016-11-23	San Francisco
104	David Brown	IT	82500.00	2020-07-10	New York
105	Eve Davis	Marketing	60000.00	2019-02-28	Los Angeles
106	Frank Miller	IT	79200.00	2021-08-16	Boston
107	Grace Lee	HR	52000.00	2017-04-05	Chicago
108	Henry Wilson	Finance	81000.00	2015-09-30	San Francisco
109	Isabel Clark	Marketing	63000.00	2022-01-12	Los Angeles
111	Laura King	Finance	85000.00	2024-04-10	Boston

3.2 DRL Queries Using Clauses, Operators, and Functions

a) Retrieve employees with salary above average salary

```
SELECT EmpID, EmpName, Dept, Salary
FROM Employee
WHERE Salary > (SELECT AVG(Salary) FROM Employee);
```

✅ Explanation:

This query finds all employees whose salary is greater than the average salary of all employees.

b) Display employees with their years of service

```
SELECT EmpID, EmpName, Dept,
    TIMESTAMPDIFF(YEAR, JoiningDate, CURDATE()) AS YearsOfService
FROM Employee;
```

✅ Explanation:

TIMESTAMPDIFF(YEAR, JoiningDate, CURDATE()) calculates how many years each employee has worked since their joining date.

c) Retrieve employees whose name starts with 'A'

```
SELECT EmpID, EmpName, Dept, City
FROM Employee
WHERE EmpName LIKE 'A%';
```

✅ Explanation:

This uses the string operator LIKE to match employees whose names start with ‘A’ (e.g., “Alice Johnson”).

d) Retrieve total salary per department

```
SELECT Dept, SUM(Salary) AS TotalSalary
FROM Employee
GROUP BY Dept;
```

✅ Explanation:

This uses the aggregate function SUM() and GROUP BY clause to find total salary by each department.

e) Retrieve employees joined in the last 2 years

```
SELECT EmpID, EmpName, Dept, JoiningDate
FROM Employee
WHERE JoiningDate >= DATE_SUB(CURDATE(), INTERVAL 2 YEAR);
```

✅ Explanation:

This retrieves employees who joined within the past 2 years from the current date using date function DATE_SUB().

f) Use CASE operator to classify employees by salary

```
SELECT EmpID, EmpName, Dept, Salary,
    CASE
      WHEN Salary >= 80000 THEN 'High Salary'
      WHEN Salary BETWEEN 60000 AND 79999 THEN 'Medium Salary'
      ELSE 'Low Salary'
    END AS SalaryCategory
FROM Employee;
```

✅ Explanation:

The CASE operator categorizes employees into salary bands.

⚙ 3.3 Set Operators Examples

Let’s assume a second table NewEmployee with similar structure.

```
CREATE TABLE NewEmployee (
 EmpID INT,
 EmpName VARCHAR(50),
 Dept VARCHAR(30),
 Salary DECIMAL(10,2),
 JoiningDate DATE,
 City VARCHAR(30)
);
```

Let’s say NewEmployee contains:

EmpID	EmpName	Dept	Salary	JoiningDate	City
109	Isabel Clark	Marketing	63000.00	2022-01-12	Los Angeles
111	Laura King	Finance	85000.00	2024-04-10	Boston
112	Michael Ross	IT	70000.00	2023-02-20	Chicago

a) Combine employees from both tables without duplicates (UNION)

```
SELECT * FROM Employee
UNION
SELECT * FROM NewEmployee;
```

✅ Explanation:

UNION merges both tables’ results, removing duplicates.

b) Find employees common in both tables (INTERSECT)

Note: MySQL doesn’t directly support INTERSECT, so use INNER JOIN as an alternative.

```
SELECT E.*
FROM Employee E
INNER JOIN NewEmployee N ON E.EmpID = N.EmpID;
```

✅ Explanation:

This finds employees present in both tables.

c) Find employees in Employee but not in NewEmployee (MINUS / EXCEPT)

MySQL alternative uses LEFT JOIN.

```
SELECT E.*
FROM Employee E
LEFT JOIN NewEmployee N ON E.EmpID = N.EmpID
WHERE N.EmpID IS NULL;
```

✅ Explanation:

This retrieves employees that exist only in Employee, not in NewEmployee.

💬 3.4 Using String Functions

a) Concatenate employee name and city

```
SELECT EmpID, CONCAT(EmpName, ' - ', City) AS EmployeeLocation
FROM Employee;
```

✅ Explanation:

CONCAT() combines employee name and city into a single string.

b) Find employees with name length greater than 6

```
SELECT EmpID, EmpName
FROM Employee
WHERE LENGTH(EmpName) > 6;
```

✅ Explanation:

LENGTH() (or CHAR_LENGTH() in some DBMS) returns the number of characters in EmpName.