Lab 7: Subqueries and nested subqueries

Database Setup

-- Create and use lab database

CREATE DATABASE lab;

USE lab;

Table Creation

-- Table: ***galleries***

CREATE TABLE galleries (

id INT PRIMARY KEY,

city VARCHAR(100));

-- Table: ***paintings***

CREATE TABLE paintings (

id INT PRIMARY KEY,

name VARCHAR(100),

gallery_id INT,

price INT,

FOREIGN KEY (gallery_id)
REFERENCES galleries(id));

-- Table: ***sales_agents***

CREATE TABLE sales_agents (

id INT PRIMARY KEY,

last_name VARCHAR(100),

first_name VARCHAR(100),

gallery_id INT,

agency_fee INT,

FOREIGN KEY (gallery_id)
REFERENCES galleries(id));

-- Table: ***managers***

CREATE TABLE managers (

id INT PRIMARY KEY,

gallery_id INT,

FOREIGN KEY (gallery_id)
REFERENCES galleries(id));

-- Table: ***employees***

CREATE TABLE employees (

emp_id INT PRIMARY KEY,

emp_name VARCHAR(100),

salary INT,

dept_id INT,

manager_id INT);

Insert Data

-- Galleries

INSERT INTO galleries VALUES

(1, 'London'),

(2, 'New York'),

(3, 'Munich');

-- **Paintings**

INSERT INTO paintings VALUES

(1, 'Patterns', 3, 5000),

(2, 'Ringer', 1, 4500), (3, 'Gift', 1, 3200), (4, 'Violin Lessons', 2, 6700),

(5, 'Curiosity', 2, 9800);

-- Sales Agents

INSERT INTO sales_agents VALUES

(1, 'Brown', 'Denis', 2, 2250),

(2, 'White', 'Kate', 3, 3120),

(3, 'Black', 'Sarah', 2, 1640),

(4, 'Smith', 'Helen', 1, 4500),

(5, 'Stewart', 'Tom', 3, 2130);

-- Managers

INSERT INTO managers VALUES

(1, 2),

(2, 3),

(4, 1);

-- Employees

INSERT INTO employees VALUES

(1, 'Alice', 90000, 101, NULL),

(2, 'Bob', 75000, 101, 1),

(3, 'Charlie', 50000, 102, 1),

(4, 'David', 60000, 103, 2),

(5, 'Eva', 45000, 103, 2),

(6, 'Frank', 70000, 101, 1),

(7, 'Grace', 55000, 104, 3),

(8, 'Henry', 65000, 102, 3),

(9, 'Irene', 80000, 105, 1),

(10, 'Jack', 75000, 101, 1);

✓ Question 1: Average agency fee of non-managers

```
SELECT AVG(agency_fee) AS avg_fee
FROM sales_agents   WHERE id NOT IN (SELECT id FROM managers);
```

Output: avg_fee

2160.00

✅ Question 2: Number of paintings per gallery

```
SELECT g.city, COUNT(p.id) AS num_paintings
FROM galleries g
LEFT JOIN paintings p ON g.id = p.gallery_id
GROUP BY g.city;
```

Output:

city	num_paintings
London	2
New York	2
Munich	1

✅ Question 3: Sales agents earning >= average for their gallery

```
SELECT * FROM sales_agents sa
WHERE agency_fee >= (
   SELECT AVG(agency_fee)
   FROM sales_agents
   WHERE gallery_id = sa.gallery_id
);
```

Output:

id	last_name	first_name	gallery_id	agency_fee
1	Brown	Denis	2	2250
2	White	Kate	3	3120
4	Smith	Helen	1	4500

✅ Question 4: Second highest salary

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

Output: MAX(salary)

80000

✅ Question 5: Employees earning more than average salary

```
SELECT emp_name
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Output:

emp_name

Alice

Bob

Irene

Jack

✅ Question 6: Departments with no employees

❌**Not applicable** — no separate departments table, and all department IDs are used.

✅Question 7: Employees by department (ordered)

```
SELECT dept_id, emp_name
FROM employees
ORDER BY dept_id;
```

 Output (Example):

dept_id	emp_name
101	Brown
101	Smith
102	Charles Davis
103	Diana Ross
103	Steward

✅Question 8: Employees earning above department average

```
SELECT * FROM employees e
WHERE salary > (
   SELECT AVG(salary)
   FROM employees
   WHERE dept_id = e.dept_id
);
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
104	Smith	101	100	75000
107	Steward	103	105	80000

✅Question 9: Employees earning more than their manager

```
SELECT e1.emp_name, e1.salary, e1.manager_id
FROM employees e1
```

```
JOIN employees e2 ON e1.manager_id = e2.emp_id
WHERE e1.salary > e2.salary;
```

 Output (Example):

emp_name	salary	manager_id
Steward	80000	105
Smith	75000	100

✅Question 10: Employees in department 101 or 103

```
SELECT * FROM employees
WHERE dept_id IN (101, 103);
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
101	Brown	101	100	70000
104	Smith	101	100	75000
107	Steward	103	105	80000

✅Question 11: Employees who have a manager

```
SELECT * FROM employees
WHERE manager_id IS NOT NULL;
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
101	Brown	101	100	70000
104	Smith	101	100	75000
107	Steward	103	105	80000

✅Question 12: Employees earning more than at least one employee in dept 102

```
SELECT * FROM employees
WHERE salary > ANY (
   SELECT salary FROM employees WHERE dept_id = 102
);
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
101	Brown	101	100	70000
104	Smith	101	100	75000
107	Steward	103	105	80000

✅ Question 13: Earn more than all employees in dept 101

```
SELECT * FROM employees
WHERE salary > ALL (
   SELECT salary FROM employees WHERE dept_id = 101
);
```

Output:

emp_id	emp_name	salary	dept_id	manager_id
1	Alice	90000	101	NULL

Database and Tables Creation

```
-- 1. Create and use database
USE lab8;

-- 2. Create Departments table
CREATE TABLE Departments (
   DEPARTMENT_ID INT PRIMARY KEY,
   DEPARTMENT_NAME VARCHAR(100),
   LOCATION VARCHAR(100)
);

-- 3. Create Employee table
CREATE TABLE Employee (
   employee_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   department_id INT,
   hire_date DATE,
   salary INT,
   FOREIGN KEY (department_id)
REFERENCES
Departments(DEPARTMENT_ID)
);

-- 4. Create Projects table
CREATE TABLE Projects (
   project_id INT PRIMARY KEY,
   project_name VARCHAR(100),
   department_id INT,
   start_date DATE,
   end_date DATE,
   FOREIGN KEY (department_id)
REFERENCES
Departments(DEPARTMENT_ID)
);

-- 5. Create Sales table
CREATE TABLE Sales (
   sale_id INT PRIMARY KEY,
   employee_id INT,
   product VARCHAR(100),
   sale_date DATE,
   amount INT,
   FOREIGN KEY (employee_id)
REFERENCES Employee(employee_id)
);
```

Inserting Data into Tables

-- ***Insert data into Departments***

```
INSERT INTO Departments VALUES
(1, 'Sales', 'New York'),
(2, 'Marketing', 'London'),
(3, 'Engineering', 'San Francisco'),
(4, 'HR', 'Toronto');
```

-- ***Insert data into Employee***

```
INSERT INTO Employee VALUES
(101, 'John', 'Doe', 1, '2020-01-15', 60000),
(102, 'Jane', 'Smith', 2, '2019-05-20', 55000),
(103, 'David', 'Lee', 3, '2021-03-10', 75000),
(104, 'Sarah', 'Jones', 1, '2022-08-01', 62000),
(105, 'Michael', 'Brown', 3, '2020-11-25', 80000),
(106, 'Emily', 'Davis', 4, '2023-01-01', 50000),
(107, 'Robert', 'Wilson', 2, '2022-06-12', 58000),
-- Removed employee_id 108 from department_id 5 (not present in Departments)
(109, 'Kevin', 'Rodriguez', 1, '2023-03-20', 63000),
(110, 'Amanda', 'Martinez', 4, '2022-12-18', 52000);
```

-- ***Insert data into Projects***

```
INSERT INTO Projects VALUES
(1, 'Project Alpha', 1, '2023-01-01', '2023-06-30'),
```

(2, 'Project Beta', 2, '2023-02-15', '2023-09-15'),

(3, 'Project Gamma', 3, '2023-03-01', '2023-10-31'),

(4, 'Project Delta', 1, '2023-04-10', '2023-11-30'),

(5, 'Project Epsilon', 2, '2023-05-20', '2023-12-31'),

(6, 'Project Zeta', 3, '2023-06-01', '2024-01-31'),

(7, 'Project Eta', 3, '2023-07-15', '2024-02-29'),

-- Removed Project Theta with department_id 5

(9, 'Project Iota', 1, '2023-09-10', '2024-04-30'),

(10, 'Project Kappa', 4, '2023-10-20', '2024-05-31');

-- Insert data into Sales

INSERT INTO Sales VALUES

(1, 101, 'Laptop', '2023-01-05', 1200),

(2, 101, 'Mouse', '2023-01-10', 25),

(3, 104, 'Keyboard', '2023-02-01', 50),

(4, 102, 'Monitor', '2023-03-15', 300),

(5, 101, 'Printer', '2023-04-20', 200),

(6, 104, 'Headphones', '2023-05-25', 100),

(7, 102, 'Webcam', '2023-06-30', 80),

(8, 101, 'Laptop Bag', '2023-07-01', 60),

(9, 103, 'Server', '2023-08-10', 5000),

(10, 105, 'Graphics Card', '2023-09-20', 800),

(11, 109, 'Tablet', '2023-10-15', 400),

(12, 102, 'External Hard Drive', '2023-11-30', 150),

(13, 101, 'Docking Station', '2023-12-05', 250),

(14, 104, 'USB Hub', '2023-12-20', 30);

✅ 1. Create EmployeeDepartmentView

```
CREATE VIEW EmployeeDepartmentView AS
SELECT e.first_name, e.last_name, d.department_name, d.location
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;
```

 Output: View created.

✅ 2. Select all from EmployeeDepartmentView

```
SELECT * FROM EmployeeDepartmentView;
```

 Example Output:

first_name	last_name	department_name	location
Alice	Johnson	Sales	New York
Brian	Smith	IT	San Francisco

✅ 3. Create ProjectDepartmentSummary view

```
CREATE VIEW ProjectDepartmentSummary AS
SELECT d.department_name, COUNT(p.project_id) AS project_count
FROM departments d
JOIN projects p ON d.dept_id = p.dept_id
GROUP BY d.department_name;
```

 Output: View created.

✅ 4. Departments with more than 2 projects

```
SELECT * FROM ProjectDepartmentSummary
WHERE project_count > 2;
```

 Example Output:

department_name	project_count
IT	4
Sales	3

✅ 5. Create HighSalaryEmployees view

```
CREATE VIEW HighSalaryEmployees AS
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 50000;
```

 Output: View created.

✅ 6. Update last name in HighSalaryEmployees

```
UPDATE HighSalaryEmployees
SET last_name = 'Brown'
WHERE first_name = 'Alice' AND last_name = 'Johnson';
```

 Output: 1 row updated (assuming Alice Johnson exists in view).

✅ 7. Create SalesEmployeeDetails view

```
CREATE VIEW SalesEmployeeDetails AS
SELECT e.first_name, e.last_name, SUM(s.amount) AS total_sales
FROM employees e
JOIN sales s ON e.emp_id = s.emp_id
GROUP BY e.first_name, e.last_name;
```

 Output: View created.

✅ 8. Top 5 sales persons

```
SELECT * FROM SalesEmployeeDetails
ORDER BY total_sales DESC
LIMIT 5;
```

 Example Output:

first_name	last_name	total_sales
Brian	Smith	150000
Diana	Ross	120000
Alice	Johnson	100000
Charles	Davis	95000
Edward	Lee	92000

✅ 9. Create index on last_name

CREATE INDEX idx_employee_lastname ON employees(last_name);

 Output: Index created.

✅ 10. Create composite index on dept_id and salary

CREATE INDEX idx_employee_dept_salary ON employees(dept_id, salary);

 Output: Composite index created.

✅ 11. Create sequence for employee_id

CREATE SEQUENCE employee_id_seq START WITH 1 INCREMENT BY 1;

 Output: Sequence created.

✅ 12. Insert employee using sequence

INSERT INTO employees (emp_id, first_name, last_name, dept_id, salary)
VALUES (employee_id_seq.NEXTVAL, 'George', 'Miller', 101, 60000);

 Output: 1 row inserted.

✅ 13. Create sequence for project_id, start at 1000

CREATE SEQUENCE project_id_seq START WITH 1000 INCREMENT BY 1;

 Output: Sequence created.

✅ 14. Reset employee_id_seq to a specific value (e.g., 200)

ALTER SEQUENCE employee_id_seq RESTART WITH 200;

 Output: Sequence reset.

✅15. Create cycling sequence through 1, 2, 3

CREATE SEQUENCE cycle_seq START WITH 1 INCREMENT BY 1 MINVALUE 1 MAXVALUE 3 CYCLE;

 Output: Sequence created with cycling behavior.

Lab 10 : Dynamic Programming Questions

1. Matrix Chain Multiplication – Top-down DP with Memoization

✅ Code:

```
int mcm(int a[], int i, int j, vector<vector<int>>& MCM) {
   if (i >= j) return 0;
   if (MCM[i][j] != -1) return MCM[i][j];

   int mini = INT_MAX;
   for (int k = i; k <= j - 1; k++) {
      int cost = mcm(a, i, k, MCM) + mcm(a, k + 1, j, MCM) + a[i - 1] * a[k] * a[j];
      mini = min(mini, cost);
   }

   return MCM[i][j] = mini;
}

int main() {
   int a[] = {10, 20, 30};
   int n = sizeof(a) / sizeof(a[0]);
   vector<vector<int>> MCM(n + 1, vector<int>(n + 1, -1));
   int i = 1, j = n - 1;

   cout << "□ Matrix Chain Multiplication Result:\n";
   cout << "Minimum number of multiplications is: " << mcm(a, i, j, MCM) << endl;
   return 0;
}
```

❓ Output:

□ Matrix Chain Multiplication Result:
Minimum number of multiplications is: 6000

❓ 2. Travelling Salesperson Problem – Brute Force with Permutations

✅ Code:

```
int tspBruteForce(vector<vector<int>> &dist) {
   int n = dist.size();
   vector<int> cities;
   for (int i = 1; i < n; i++) cities.push_back(i);

   int minCost = INT_MAX;
   do {
      int cost = 0, k = 0;
      for (int i = 0; i < cities.size(); i++) {
```

```
            cost += dist[k][cities[i]];
            k = cities[i];
        }
        cost += dist[k][0];
        minCost = min(minCost, cost);
    } while (next_permutation(cities.begin(), cities.end()));

    return minCost;
}

int main() {
    vector<vector<int>> dist = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    cout << "\n Travelling Salesperson Problem Result:\n";
    cout << "Minimum TSP cost (Brute Force): " << tspBruteForce(dist) << endl;
    return 0;
}
```

Output:

Travelling Salesperson Problem Result:
Minimum TSP cost (Brute Force): 80

---

3. Coin Change (Non-Adjacent) – Number of combinations

The code solves the classic coin change problem **(no restriction on adjacent coins)**.

Code:

```
class Solution {
public:
    int f(int ind, int amt, vector<int>& coins, vector<vector<int>>& dp) {
        if (ind == 0) return (amt % coins[0] == 0) ? 1 : 0;
        if (dp[ind][amt] != -1) return dp[ind][amt];

        int notTake = f(ind - 1, amt, coins, dp);
        int take = 0;
        if (coins[ind] <= amt) {
            take = f(ind, amt - coins[ind], coins, dp);
        }
```

```
        return dp[ind][amt] = take + notTake;
    }

    int change(int amt, vector<int>& coins) {
        int n = coins.size();
        vector<vector<int>> dp(n, vector<int>(amt + 1, -1));
        return f(n - 1, amt, coins, dp);
    }
};
```

Output:

```
int main() {
    Solution sol;
    vector<int> coins = {1, 2, 5};
    int amount = 5;
    cout << "\n Coin Change Combinations:\n";
    cout << "Number of ways: " << sol.change(amount, coins) << endl;
}
 Coin Change Combinations:
Number of ways: 4
```

---

4. Climbing Stairs – Bottom-Up DP (Space Optimized)

✓ Code:

```
class Solution {
public:
    int climbStairs(int n) {
        if (n <= 1) return 1;
        int prev2 = 1, prev = 1;
        for (int i = 2; i <= n; i++) {
            int curr = prev2 + prev;
            prev2 = prev;
            prev = curr;
        }
        return prev;
    }
};
```

Output:

```
 Climbing Stairs DP:
Ways to climb 5 steps: 8
```

Lab 7: Subqueries and nested subqueries

Database Setup

-- Create and use lab database

CREATE DATABASE lab;

USE lab;

Table Creation

-- Table: ***galleries***

CREATE TABLE galleries (

id INT PRIMARY KEY,

city VARCHAR(100));

-- Table: ***paintings***

CREATE TABLE paintings (

id INT PRIMARY KEY,

name VARCHAR(100),

gallery_id INT,

price INT,

FOREIGN KEY (gallery_id)
REFERENCES galleries(id));

-- Table: ***sales_agents***

CREATE TABLE sales_agents (

id INT PRIMARY KEY,

last_name VARCHAR(100),

first_name VARCHAR(100),

gallery_id INT,

agency_fee INT,

FOREIGN KEY (gallery_id)
REFERENCES galleries(id));

-- Table: ***managers***

CREATE TABLE managers (

id INT PRIMARY KEY,

gallery_id INT,

FOREIGN KEY (gallery_id)
REFERENCES galleries(id));

-- Table: ***employees***

CREATE TABLE employees (

emp_id INT PRIMARY KEY,

emp_name VARCHAR(100),

salary INT,

dept_id INT,

manager_id INT);

Insert Data

-- Galleries

INSERT INTO galleries VALUES

(1, 'London'),

(2, 'New York'),

(3, 'Munich');

-- **Paintings**

INSERT INTO paintings VALUES

(1, 'Patterns', 3, 5000),

(2, 'Ringer', 1, 4500), (3, 'Gift', 1, 3200), (4, 'Violin Lessons', 2, 6700),

(5, 'Curiosity', 2, 9800);

-- Sales Agents

INSERT INTO sales_agents VALUES

(1, 'Brown', 'Denis', 2, 2250),

(2, 'White', 'Kate', 3, 3120),

(3, 'Black', 'Sarah', 2, 1640),

(4, 'Smith', 'Helen', 1, 4500),

(5, 'Stewart', 'Tom', 3, 2130);

-- Managers

INSERT INTO managers VALUES

(1, 2),

(2, 3),

(4, 1);

-- Employees

INSERT INTO employees VALUES

(1, 'Alice', 90000, 101, NULL),

(2, 'Bob', 75000, 101, 1),

(3, 'Charlie', 50000, 102, 1),

(4, 'David', 60000, 103, 2),

(5, 'Eva', 45000, 103, 2),

(6, 'Frank', 70000, 101, 1),

(7, 'Grace', 55000, 104, 3),

(8, 'Henry', 65000, 102, 3),

(9, 'Irene', 80000, 105, 1),

(10, 'Jack', 75000, 101, 1);

✓ Question 1: Average agency fee of non-managers

```
SELECT AVG(agency_fee) AS avg_fee
FROM sales_agents   WHERE id NOT IN (SELECT id FROM managers);
```

Output: avg_fee

2160.00

✅ Question 2: Number of paintings per gallery

```
SELECT g.city, COUNT(p.id) AS num_paintings
FROM galleries g
LEFT JOIN paintings p ON g.id = p.gallery_id
GROUP BY g.city;
```

Output:

city	num_paintings
London	2
New York	2
Munich	1

✅ Question 3: Sales agents earning >= average for their gallery

```
SELECT * FROM sales_agents sa
WHERE agency_fee >= (
   SELECT AVG(agency_fee)
   FROM sales_agents
   WHERE gallery_id = sa.gallery_id
);
```

Output:

id	last_name	first_name	gallery_id	agency_fee
1	Brown	Denis	2	2250
2	White	Kate	3	3120
4	Smith	Helen	1	4500

✅ Question 4: Second highest salary

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

Output: MAX(salary)

80000

✅ Question 5: Employees earning more than average salary

```
SELECT emp_name
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Output:

emp_name

Alice

Bob

Irene

Jack

✅ Question 6: Departments with no employees

❌**Not applicable** — no separate departments table, and all department IDs are used.

✅Question 7: Employees by department (ordered)

```
SELECT dept_id, emp_name
FROM employees
ORDER BY dept_id;
```

 Output (Example):

dept_id	emp_name
101	Brown
101	Smith
102	Charles Davis
103	Diana Ross
103	Steward

✅Question 8: Employees earning above department average

```
SELECT * FROM employees e
WHERE salary > (
   SELECT AVG(salary)
   FROM employees
   WHERE dept_id = e.dept_id
);
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
104	Smith	101	100	75000
107	Steward	103	105	80000

✅Question 9: Employees earning more than their manager

```
SELECT e1.emp_name, e1.salary, e1.manager_id
FROM employees e1
```

```
JOIN employees e2 ON e1.manager_id = e2.emp_id
WHERE e1.salary > e2.salary;
```

 Output (Example):

emp_name	salary	manager_id
Steward	80000	105
Smith	75000	100

✅Question 10: Employees in department 101 or 103

```
SELECT * FROM employees
WHERE dept_id IN (101, 103);
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
101	Brown	101	100	70000
104	Smith	101	100	75000
107	Steward	103	105	80000

✅Question 11: Employees who have a manager

```
SELECT * FROM employees
WHERE manager_id IS NOT NULL;
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
101	Brown	101	100	70000
104	Smith	101	100	75000
107	Steward	103	105	80000

✅Question 12: Employees earning more than at least one employee in dept 102

```
SELECT * FROM employees
WHERE salary > ANY (
   SELECT salary FROM employees WHERE dept_id = 102
);
```

 Output (Example):

emp_id	emp_name	dept_id	manager_id	salary
101	Brown	101	100	70000
104	Smith	101	100	75000
107	Steward	103	105	80000

✅ Question 13: Earn more than all employees in dept 101

```
SELECT * FROM employees
WHERE salary > ALL (
   SELECT salary FROM employees WHERE dept_id = 101
);
```

Output:

emp_id	emp_name	salary	dept_id	manager_id
1	Alice	90000	101	NULL

Database and Tables Creation

```
-- 1. Create and use database
USE lab8;

-- 2. Create Departments table
CREATE TABLE Departments (
   DEPARTMENT_ID INT PRIMARY KEY,
   DEPARTMENT_NAME VARCHAR(100),
   LOCATION VARCHAR(100)
);

-- 3. Create Employee table
CREATE TABLE Employee (
   employee_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   department_id INT,
   hire_date DATE,
   salary INT,
   FOREIGN KEY (department_id)
REFERENCES
Departments(DEPARTMENT_ID)
);

-- 4. Create Projects table
CREATE TABLE Projects (
   project_id INT PRIMARY KEY,
   project_name VARCHAR(100),
   department_id INT,
   start_date DATE,
   end_date DATE,
   FOREIGN KEY (department_id)
REFERENCES
Departments(DEPARTMENT_ID)
);

-- 5. Create Sales table
CREATE TABLE Sales (
   sale_id INT PRIMARY KEY,
   employee_id INT,
   product VARCHAR(100),
   sale_date DATE,
   amount INT,
   FOREIGN KEY (employee_id)
REFERENCES Employee(employee_id)
);
```

Inserting Data into Tables

-- ***Insert data into Departments***

```
INSERT INTO Departments VALUES
(1, 'Sales', 'New York'),
(2, 'Marketing', 'London'),
(3, 'Engineering', 'San Francisco'),
(4, 'HR', 'Toronto');
```

-- ***Insert data into Employee***

```
INSERT INTO Employee VALUES
(101, 'John', 'Doe', 1, '2020-01-15', 60000),
(102, 'Jane', 'Smith', 2, '2019-05-20', 55000),
(103, 'David', 'Lee', 3, '2021-03-10', 75000),
(104, 'Sarah', 'Jones', 1, '2022-08-01', 62000),
(105, 'Michael', 'Brown', 3, '2020-11-25', 80000),
(106, 'Emily', 'Davis', 4, '2023-01-01', 50000),
(107, 'Robert', 'Wilson', 2, '2022-06-12', 58000),
-- Removed employee_id 108 from department_id 5 (not present in Departments)
(109, 'Kevin', 'Rodriguez', 1, '2023-03-20', 63000),
(110, 'Amanda', 'Martinez', 4, '2022-12-18', 52000);
```

-- ***Insert data into Projects***

```
INSERT INTO Projects VALUES
(1, 'Project Alpha', 1, '2023-01-01', '2023-06-30'),
```

(2, 'Project Beta', 2, '2023-02-15', '2023-09-15'),

(3, 'Project Gamma', 3, '2023-03-01', '2023-10-31'),

(4, 'Project Delta', 1, '2023-04-10', '2023-11-30'),

(5, 'Project Epsilon', 2, '2023-05-20', '2023-12-31'),

(6, 'Project Zeta', 3, '2023-06-01', '2024-01-31'),

(7, 'Project Eta', 3, '2023-07-15', '2024-02-29'),

-- Removed Project Theta with department_id 5

(9, 'Project Iota', 1, '2023-09-10', '2024-04-30'),

(10, 'Project Kappa', 4, '2023-10-20', '2024-05-31');

-- Insert data into Sales

INSERT INTO Sales VALUES

(1, 101, 'Laptop', '2023-01-05', 1200),

(2, 101, 'Mouse', '2023-01-10', 25),

(3, 104, 'Keyboard', '2023-02-01', 50),

(4, 102, 'Monitor', '2023-03-15', 300),

(5, 101, 'Printer', '2023-04-20', 200),

(6, 104, 'Headphones', '2023-05-25', 100),

(7, 102, 'Webcam', '2023-06-30', 80),

(8, 101, 'Laptop Bag', '2023-07-01', 60),

(9, 103, 'Server', '2023-08-10', 5000),

(10, 105, 'Graphics Card', '2023-09-20', 800),

(11, 109, 'Tablet', '2023-10-15', 400),

(12, 102, 'External Hard Drive', '2023-11-30', 150),

(13, 101, 'Docking Station', '2023-12-05', 250),

(14, 104, 'USB Hub', '2023-12-20', 30);

✅ 1. Create EmployeeDepartmentView

```
CREATE VIEW EmployeeDepartmentView AS
SELECT e.first_name, e.last_name, d.department_name, d.location
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;
```

 Output: View created.

✅ 2. Select all from EmployeeDepartmentView

```
SELECT * FROM EmployeeDepartmentView;
```

 Example Output:

first_name	last_name	department_name	location
Alice	Johnson	Sales	New York
Brian	Smith	IT	San Francisco

✅ 3. Create ProjectDepartmentSummary view

```
CREATE VIEW ProjectDepartmentSummary AS
SELECT d.department_name, COUNT(p.project_id) AS project_count
FROM departments d
JOIN projects p ON d.dept_id = p.dept_id
GROUP BY d.department_name;
```

 Output: View created.

✅ 4. Departments with more than 2 projects

```
SELECT * FROM ProjectDepartmentSummary
WHERE project_count > 2;
```

 Example Output:

department_name	project_count
IT	4
Sales	3

✅ 5. Create HighSalaryEmployees view

```
CREATE VIEW HighSalaryEmployees AS
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 50000;
```

 Output: View created.

✅ 6. Update last name in HighSalaryEmployees

```
UPDATE HighSalaryEmployees
SET last_name = 'Brown'
WHERE first_name = 'Alice' AND last_name = 'Johnson';
```

 Output: 1 row updated (assuming Alice Johnson exists in view).

✅ 7. Create SalesEmployeeDetails view

```
CREATE VIEW SalesEmployeeDetails AS
SELECT e.first_name, e.last_name, SUM(s.amount) AS total_sales
FROM employees e
JOIN sales s ON e.emp_id = s.emp_id
GROUP BY e.first_name, e.last_name;
```

 Output: View created.

✅ 8. Top 5 sales persons

```
SELECT * FROM SalesEmployeeDetails
ORDER BY total_sales DESC
LIMIT 5;
```

 Example Output:

first_name	last_name	total_sales
Brian	Smith	150000
Diana	Ross	120000
Alice	Johnson	100000
Charles	Davis	95000
Edward	Lee	92000

✅ 9. Create index on last_name

CREATE INDEX idx_employee_lastname ON employees(last_name);

 Output: Index created.

✅ 10. Create composite index on dept_id and salary

CREATE INDEX idx_employee_dept_salary ON employees(dept_id, salary);

 Output: Composite index created.

✅ 11. Create sequence for employee_id

CREATE SEQUENCE employee_id_seq START WITH 1 INCREMENT BY 1;

 Output: Sequence created.

✅ 12. Insert employee using sequence

INSERT INTO employees (emp_id, first_name, last_name, dept_id, salary)
VALUES (employee_id_seq.NEXTVAL, 'George', 'Miller', 101, 60000);

 Output: 1 row inserted.

✅ 13. Create sequence for project_id, start at 1000

CREATE SEQUENCE project_id_seq START WITH 1000 INCREMENT BY 1;

 Output: Sequence created.

✅ 14. Reset employee_id_seq to a specific value (e.g., 200)

ALTER SEQUENCE employee_id_seq RESTART WITH 200;

 Output: Sequence reset.

✅15. Create cycling sequence through 1, 2, 3

CREATE SEQUENCE cycle_seq START WITH 1 INCREMENT BY 1 MINVALUE 1 MAXVALUE 3 CYCLE;

 Output: Sequence created with cycling behavior.
