Laboratory Work 6: SQL JOINs

Objective

To understand and practice different types of SQL JOIN operations including CROSS JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN, and to learn the differences between ON and WHERE clauses in join operations.

Prerequisites

- Basic SQL knowledge (SELECT, FROM, WHERE)
- Understanding of relational database concepts
- Access to a SQL database system (PostgreSQL, MySQL, or similar)

Theoretical Background

What is a JOIN?

A joined table is a table derived from two other tables according to the rules of the particular join type. JOINs allow you to combine data from multiple tables based on related columns.

Types of JOINs

1. **CROSS JOIN**: Produces a Cartesian product of two tables (all possible combinations)
2. **INNER JOIN**: Returns only matching rows from both tables
3. **LEFT JOIN (LEFT OUTER JOIN)**: Returns all rows from the left table and matching rows from the right table
4. **RIGHT JOIN (RIGHT OUTER JOIN)**: Returns all rows from the right table and matching rows from the left table
5. **FULL JOIN (FULL OUTER JOIN)**: Returns all rows from both tables, with NULLs where there's no match

Syntax Variations

- **ON clause**: Specifies the join condition
- **USING clause**: Simplified syntax when join columns have the same name
- **NATURAL JOIN**: Automatically joins on all columns with the same names

Part 1: Database Setup

Step 1.1: Create Sample Tables

Execute the following SQL commands to create the tables for this lab:

```
-- Create table: employees
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    dept_id INT,
    salary DECIMAL(10, 2)
);

-- Create table: departments
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50),
    location VARCHAR(50)
);

-- Create table: projects
CREATE TABLE projects (
    project_id INT PRIMARY KEY,
    project_name VARCHAR(50),
    dept_id INT,
    budget DECIMAL(10, 2)
);
```

Step 1.2: Insert Sample Data

```
-- Insert data into employees
INSERT INTO employees (emp_id, emp_name, dept_id, salary)
VALUES
(1, 'John Smith', 101, 50000),
(2, 'Jane Doe', 102, 60000),
(3, 'Mike Johnson', 101, 55000),
(4, 'Sarah Williams', 103, 65000),
(5, 'Tom Brown', NULL, 45000);

-- Insert data into departments
INSERT INTO departments (dept_id, dept_name, location) VALUES
(101, 'IT', 'Building A'),
(102, 'HR', 'Building B'),
(103, 'Finance', 'Building C'),
(104, 'Marketing', 'Building D');

-- Insert data into projects
INSERT INTO projects (project_id, project_name, dept_id,
budget) VALUES
(1, 'Website Redesign', 101, 100000),
(2, 'Employee Training', 102, 50000),
(3, 'Budget Analysis', 103, 75000),
(4, 'Cloud Migration', 101, 150000),
(5, 'AI Research', NULL, 200000);
```

Part 2: CROSS JOIN Exercises

Exercise 2.1: Basic CROSS JOIN

Write a query using CROSS JOIN to show all possible combinations of employees and departments.

```
-- Your query here
SELECT e.emp_name, d.dept_name
FROM employees e CROSS JOIN departments d;
```

Question: How many rows does the result contain? Calculate N × M where N = number of employees, M = number of departments.

Exercise 2.2: Alternative CROSS JOIN Syntax

Rewrite the above query using: a) Comma notation: `FROM employees, departments` b) INNER JOIN with TRUE condition: `INNER JOIN departments ON TRUE`

Exercise 2.3: Practical CROSS JOIN

Create a schedule showing all employees paired with all projects (useful for availability matrix).

Part 3: INNER JOIN Exercises

Exercise 3.1: Basic INNER JOIN with ON

Write a query to display employees with their department names (only employees who have a department).

```
-- Your query here
SELECT e.emp_name, d.dept_name, d.location
FROM employees e
INNER JOIN departments d ON e.dept_id = d.dept_id;
```

Question: How many rows are returned? Why is Tom Brown not included?

Exercise 3.2: INNER JOIN with USING

Rewrite the query from Exercise 3.1 using the USING clause instead of ON.

```
-- Your query here
SELECT emp_name, dept_name, location
FROM employees
INNER JOIN departments USING (dept_id);
```

Question: What's the difference in output columns compared to the ON version?

Exercise 3.3: NATURAL INNER JOIN

Rewrite the query using NATURAL INNER JOIN.

```
-- Your query here
SELECT emp_name, dept_name, location
FROM employees
NATURAL INNER JOIN departments;
```

Exercise 3.4: Multi-table INNER JOIN

Join all three tables to show: employee name, department name, and project name.

```
-- Your query here
SELECT e.emp_name, d.dept_name, p.project_name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.dept_id
INNER JOIN projects p ON d.dept_id = p.dept_id;
```

Part 4: LEFT JOIN Exercises

Exercise 4.1: Basic LEFT JOIN

Write a query to show all employees with their department information, including employees without a department.

```
-- Your query here
SELECT e.emp_name, e.dept_id AS emp_dept, d.dept_id AS
dept_dept, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id;
```

Question: How is Tom Brown represented in the results?

Exercise 4.2: LEFT JOIN with USING

Rewrite Exercise 4.1 using the USING clause.

Exercise 4.3: Find Unmatched Records

Write a query to find employees who are NOT assigned to any department.

```
-- Your query here
SELECT e.emp_name, e.dept_id
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id
WHERE d.dept_id IS NULL;
```

Exercise 4.4: LEFT JOIN with Aggregation

Show all departments with the count of employees in each department (include departments with 0 employees).

```
-- Your query here
SELECT d.dept_name, COUNT(e.emp_id) AS employee_count
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_id, d.dept_name
ORDER BY employee_count DESC;
```

Part 5: RIGHT JOIN Exercises

Exercise 5.1: Basic RIGHT JOIN

Write a query to show all departments with their employees, including departments without employees.

```
-- Your query here
SELECT e.emp_name, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.dept_id;
```

Exercise 5.2: Convert to LEFT JOIN

Rewrite Exercise 5.1 using LEFT JOIN instead of RIGHT JOIN (hint: reverse the table order).

Exercise 5.3: Find Departments Without Employees

Write a query to identify departments that have no employees assigned.

```
-- Your query here
SELECT d.dept_name, d.location
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.dept_id
WHERE e.emp_id IS NULL;
```

Part 6: FULL JOIN Exercises

Exercise 6.1: Basic FULL JOIN

Write a query to show all employees and all departments, with NULL values where there's no match.

```
-- Your query here
SELECT e.emp_name, e.dept_id AS emp_dept, d.dept_id AS
dept_dept, d.dept_name
FROM employees e
FULL JOIN departments d ON e.dept_id = d.dept_id;
```

Question: Which records have NULL values on the left side? Which have NULL on the right side?

Exercise 6.2: FULL JOIN with Projects

Show all departments and all projects, including those without matches.

```
-- Your query here
SELECT d.dept_name, p.project_name, p.budget
FROM departments d
FULL JOIN projects p ON d.dept_id = p.dept_id;
```

Exercise 6.3: Find Orphaned Records

Using FULL JOIN, write a query to find both:

- Employees without departments
- Departments without employees

```
-- Your query here
SELECT
    CASE
        WHEN e.emp_id IS NULL THEN 'Department without
employees'
        WHEN d.dept_id IS NULL THEN 'Employee without
department'
        ELSE 'Matched'
    END AS record_status,
    e.emp_name,
    d.dept_name
FROM employees e
FULL JOIN departments d ON e.dept_id = d.dept_id
WHERE e.emp_id IS NULL OR d.dept_id IS NULL;
```

Part 7: ON vs WHERE Clause

Exercise 7.1: Filtering in ON Clause (Outer Join)

Write a query using LEFT JOIN with an additional condition in the ON clause.

```
-- Query 1: Filter in ON clause
SELECT e.emp_name, d.dept_name, e.salary
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id AND
d.location = 'Building A';
```

Exercise 7.2: Filtering in WHERE Clause (Outer Join)

Write the same query but move the location filter to the WHERE clause.

```
-- Query 2: Filter in WHERE clause
```

```
SELECT e.emp_name, d.dept_name, e.salary
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id
WHERE d.location = 'Building A';
```

Question: Compare the results of Query 1 and Query 2. Explain the difference.

Answer:

- Query 1 (ON clause): Applies the filter BEFORE the join, so all employees are included, but only departments in Building A are matched.
- Query 2 (WHERE clause): Applies the filter AFTER the join, so employees are excluded if their department is not in Building A.

Exercise 7.3: ON vs WHERE with INNER JOIN

Repeat exercises 7.1 and 7.2 using INNER JOIN instead of LEFT JOIN.

Question: Is there any difference in results? Why or why not?

Part 8: Complex JOIN Scenarios

Exercise 8.1: Multiple Joins with Different Types

Write a query that combines different join types:

- Show all departments
- Include employee information (if any)
- Include project information (if any)

```
-- Your query here
SELECT
    d.dept_name,
    e.emp_name,
    e.salary,
    p.project_name,
    p.budget
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
LEFT JOIN projects p ON d.dept_id = p.dept_id
ORDER BY d.dept_name, e.emp_name;
```

Exercise 8.2: Self Join

Add a manager_id column to employees table and write a self-join query to show employees with their managers.

```
-- Add manager_id column
ALTER TABLE employees ADD COLUMN manager_id INT;

-- Update with sample data
```

```
UPDATE employees SET manager_id = 3 WHERE emp_id = 1;
UPDATE employees SET manager_id = 3 WHERE emp_id = 2;
UPDATE employees SET manager_id = NULL WHERE emp_id = 3;
UPDATE employees SET manager_id = 3 WHERE emp_id = 4;
UPDATE employees SET manager_id = 3 WHERE emp_id = 5;

-- Self join query
SELECT
    e.emp_name AS employee,
    m.emp_name AS manager
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.emp_id;
```

Exercise 8.3: Join with Subquery

Find departments where the average employee salary is above $50,000.

```
-- Your query here
SELECT d.dept_name, AVG(e.salary) AS avg_salary
FROM departments d
INNER JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_id, d.dept_name
HAVING AVG(e.salary) > 50000;
```

Lab Questions

1. What is the difference between INNER JOIN and LEFT JOIN?
2. When would you use CROSS JOIN in a practical scenario?
3. Explain why the position of a filter condition (ON vs WHERE) matters for outer joins but not for inner joins.
4. What is the result of: `SELECT COUNT(*) FROM table1 CROSS JOIN table2` if table1 has 5 rows and table2 has 10 rows?
5. How does NATURAL JOIN determine which columns to join on?
6. What are the potential risks of using NATURAL JOIN?
7. Convert this LEFT JOIN to a RIGHT JOIN: `SELECT * FROM A LEFT JOIN B ON A.id = B.id`
8. When should you use FULL OUTER JOIN instead of other join types?

Deliverables

Submit a document containing:

1. All SQL queries from exercises

Additional Challenges (Optional)

1. Create a query that simulates FULL OUTER JOIN using UNION of LEFT and RIGHT joins (for databases that don't support FULL OUTER JOIN).
2. Write a query to find employees who work in departments that have more than one project.
3. Create a hierarchical query showing the complete organizational structure using self-joins (employee → manager → manager's manager, etc.).
4. Implement a query that finds all pairs of employees who work in the same department.

References

- SQL Standard Documentation
- PostgreSQL JOIN Documentation: https://www.postgresql.org/docs/current/queries-table-expressions.html
- Visual representation of SQL JOINs

Notes

- Save all your queries in a .sql file
- Test each query before submitting
- Pay attention to NULL handling in outer joins
- Use meaningful aliases for tables
- Comment your complex queries

Good luck!