**Setup: Sample Tables**

To practice these exercises, use the following tables:

**Table: employees**

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

department\_id INT,

salary DECIMAL(10,2)

);

INSERT INTO employees (employee\_id, first\_name, last\_name, department\_id, salary) VALUES

(1, 'John', 'Doe', 1, 60000),

(2, 'Jane', 'Smith', 2, 75000),

(3, 'Mike', 'Brown', 1, 50000),

(4, 'Emily', 'Davis', 3, 80000),

(5, 'David', 'Wilson', NULL, 65000);

**Table: departments**

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50)

);

INSERT INTO departments (department\_id, department\_name) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

**INNER JOIN Exercises**

**1. Basic INNER JOIN**

**Exercise:** Retrieve all employees and their corresponding department names.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id;

**2. INNER JOIN with a Condition**

**Exercise:** Retrieve employees who work in the "Finance" department.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

WHERE d.department\_name = 'Finance';

**3. INNER JOIN with Aggregate Function**

**Exercise:** Find the total salary per department.

**Solution:**

SELECT d.department\_name, SUM(e.salary) AS total\_salary

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

GROUP BY d.department\_name;

**LEFT JOIN Exercises**

**4. Basic LEFT JOIN**

**Exercise:** Retrieve all employees, including those without a department.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id;

**5. LEFT JOIN with Filtering**

**Exercise:** Retrieve employees who do not belong to any department.

**Solution:**

SELECT e.first\_name, e.last\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id

WHERE d.department\_id IS NULL;

**RIGHT JOIN Exercises**

**6. Basic RIGHT JOIN**

**Exercise:** Retrieve all departments, including those without employees.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.department\_id;

**7. RIGHT JOIN with COUNT**

**Exercise:** Count the number of employees in each department, including empty ones.

**Solution:**

SELECT d.department\_name, COUNT(e.employee\_id) AS employee\_count

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.department\_id

GROUP BY d.department\_name;

**FULL OUTER JOIN (Using UNION)**

**8. Simulating FULL OUTER JOIN**

**Exercise:** Retrieve all employees and departments, including unmatched records.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id

UNION

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.department\_id;

**CROSS JOIN Exercises**

**9. Basic CROSS JOIN**

**Exercise:** Pair each employee with every department.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

CROSS JOIN departments d;

**10. CROSS JOIN with a WHERE Clause**

**Exercise:** Find all possible department transfers for employees.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name AS possible\_transfer\_department

FROM employees e

CROSS JOIN departments d

WHERE e.department\_id <> d.department\_id;

**SELF JOIN Exercises**

**11. Basic SELF JOIN**

**Exercise:** Find employees with the same department.

**Solution:**

SELECT e1.first\_name AS emp1, e2.first\_name AS emp2, d.department\_name

FROM employees e1

JOIN employees e2 ON e1.department\_id = e2.department\_id AND e1.employee\_id <> e2.employee\_id

JOIN departments d ON e1.department\_id = d.department\_id;

**12. Manager-Employee Relationship (Self JOIN)**

**Exercise:** Assume employees has a manager\_id. Retrieve employees and their managers.

**Solution:**

SELECT e.first\_name AS employee, m.first\_name AS manager

FROM employees e

LEFT JOIN employees m ON e.manager\_id = m.employee\_id;

**JOIN with Multiple Tables**

**13. Joining 3 Tables**

**Exercise:** Retrieve employees, their departments, and managers.

**Solution:**

SELECT e.first\_name AS employee, d.department\_name, m.first\_name AS manager

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

LEFT JOIN employees m ON e.manager\_id = m.employee\_id;

**14. INNER JOIN with Subquery**

**Exercise:** Retrieve employees who earn above the average salary in their department.

**Solution:**

SELECT e.first\_name, e.salary, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

WHERE e.salary > (SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id);

**15. LEFT JOIN with COALESCE**

**Exercise:** Show department names and employee count, displaying "No Employees" if empty.

**Solution:**

SELECT d.department\_name, COALESCE(COUNT(e.employee\_id), 'No Employees') AS employee\_count

FROM departments d

LEFT JOIN employees e ON d.department\_id = e.department\_id

GROUP BY d.department\_name;

**16. RIGHT JOIN with COALESCE**

**Exercise:** Retrieve employees and their department names, displaying "No Department" if NULL.

**Solution:**

SELECT e.first\_name, COALESCE(d.department\_name, 'No Department') AS department

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.department\_id;

**17. Filtering with HAVING and JOIN**

**Exercise:** Find departments with more than one employee.

**Solution:**

SELECT d.department\_name, COUNT(e.employee\_id) AS employee\_count

FROM departments d

JOIN employees e ON d.department\_id = e.department\_id

GROUP BY d.department\_name

HAVING employee\_count > 1;

**18. Using JOIN with DELETE**

**Exercise:** Delete employees from departments that no longer exist.

**Solution:**

DELETE e FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id

WHERE d.department\_id IS NULL;

**19. Using JOIN with UPDATE**

**Exercise:** Increase salary by 10% for employees in the "IT" department.

**Solution:**

UPDATE employees e

JOIN departments d ON e.department\_id = d.department\_id

SET e.salary = e.salary \* 1.10

WHERE d.department\_name = 'IT';

**20. JOIN with ORDER BY**

**Exercise:** Retrieve employees sorted by department name.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

ORDER BY d.department\_name;

**Setup: Sample Tables**

To practice these exercises, use the following tables:

**Table: employees**

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

department\_id INT,

salary DECIMAL(10,2),

hire\_date DATE

);

INSERT INTO employees (employee\_id, first\_name, last\_name, department\_id, salary, hire\_date) VALUES

(1, 'John', 'Doe', 1, 60000, '2018-06-10'),

(2, 'Jane', 'Smith', 2, 75000, '2017-03-15'),

(3, 'Mike', 'Brown', 1, 50000, '2020-11-20'),

(4, 'Emily', 'Davis', 3, 80000, '2016-07-25'),

(5, 'David', 'Wilson', NULL, 65000, '2019-09-30');

**Table: departments**

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50)

);

INSERT INTO departments (department\_id, department\_name) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

**Single-Row Subqueries**

**1. Subquery in WHERE Clause**

**Exercise:** Retrieve employees who earn more than the average salary.

**Solution:**

SELECT first\_name, last\_name, salary

FROM employees

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

**2. Subquery with a Specific Value**

**Exercise:** Find employees who work in the department where ‘Jane Smith’ works.

**Solution:**

SELECT first\_name, last\_name

FROM employees

WHERE department\_id = (SELECT department\_id FROM employees WHERE first\_name = 'Jane' AND last\_name = 'Smith');

**3. Subquery in SELECT Clause**

**Exercise:** Retrieve employees and their department names.

**Solution:**

SELECT first\_name, last\_name,

(SELECT department\_name FROM departments WHERE department\_id = employees.department\_id) AS department\_name

FROM employees;

**Multi-Row Subqueries**

**4. Subquery with IN Operator**

**Exercise:** Retrieve employees who work in departments with at least one employee earning more than 70,000.

**Solution:**

SELECT first\_name, last\_name

FROM employees

WHERE department\_id IN (SELECT department\_id FROM employees WHERE salary > 70000);

**5. Subquery with NOT IN**

**Exercise:** Retrieve employees who are not assigned to any department.

**Solution:**

SELECT first\_name, last\_name

FROM employees

WHERE department\_id NOT IN (SELECT department\_id FROM departments);

**6. Subquery with EXISTS**

**Exercise:** Retrieve employees working in a department that has employees.

**Solution:**

SELECT first\_name, last\_name

FROM employees e

WHERE EXISTS (SELECT 1 FROM departments d WHERE e.department\_id = d.department\_id);

**7. Subquery with NOT EXISTS**

**Exercise:** Retrieve employees who do not belong to any department.

**Solution:**

SELECT first\_name, last\_name

FROM employees e

WHERE NOT EXISTS (SELECT 1 FROM departments d WHERE e.department\_id = d.department\_id);

**Correlated Subqueries**

**8. Correlated Subquery in WHERE**

**Exercise:** Retrieve employees who earn more than the average salary in their department.

**Solution:**

SELECT first\_name, last\_name, salary

FROM employees e1

WHERE salary > (SELECT AVG(salary) FROM employees e2 WHERE e1.department\_id = e2.department\_id);

**9. Correlated Subquery in SELECT**

**Exercise:** Retrieve employees and show their department’s average salary.

**Solution:**

SELECT first\_name, last\_name, salary,

(SELECT AVG(salary) FROM employees e2 WHERE e1.department\_id = e2.department\_id) AS dept\_avg\_salary

FROM employees e1;

**10. Correlated Subquery in FROM Clause**

**Exercise:** Retrieve the highest-paid employee in each department.

**Solution:**

SELECT e.first\_name, e.last\_name, e.salary, d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

WHERE salary = (SELECT MAX(salary) FROM employees WHERE department\_id = e.department\_id);

**Subqueries with Aggregate Functions**

**11. Using COUNT with Subquery**

**Exercise:** Retrieve departments with more than one employee.

**Solution:**

SELECT department\_id, department\_name

FROM departments

WHERE department\_id IN (SELECT department\_id FROM employees GROUP BY department\_id HAVING COUNT(\*) > 1);

**12. Subquery in HAVING Clause**

**Exercise:** Find departments where the total salary exceeds 100,000.

**Solution:**

SELECT department\_id, SUM(salary) AS total\_salary

FROM employees

GROUP BY department\_id

HAVING SUM(salary) > (SELECT AVG(salary) \* COUNT(\*) FROM employees);

**13. Subquery in UPDATE Statement**

**Exercise:** Increase the salary of employees earning less than their department’s average salary.

**Solution:**

UPDATE employees e

SET salary = salary \* 1.10

WHERE salary < (SELECT AVG(salary) FROM employees e2 WHERE e.department\_id = e2.department\_id);

**14. Subquery in DELETE Statement**

**Exercise:** Delete employees who have the lowest salary in their department.

**Solution:**

DELETE FROM employees

WHERE salary = (SELECT MIN(salary) FROM (SELECT \* FROM employees) AS sub WHERE department\_id = employees.department\_id);

**Advanced Subquery Scenarios**

**15. Using LIMIT with Subquery**

**Exercise:** Retrieve the second highest salary.

**Solution:**

SELECT salary FROM employees

ORDER BY salary DESC

LIMIT 1 OFFSET 1;

**16. Using EXISTS with Multiple Conditions**

**Exercise:** Retrieve employees who work in departments that have employees with a salary above 75,000.

**Solution:**

SELECT first\_name, last\_name

FROM employees e

WHERE EXISTS (SELECT 1 FROM employees e2 WHERE e.department\_id = e2.department\_id AND e2.salary > 75000);

**17. Using Subquery in CASE Statement**

**Exercise:** Categorize employees based on whether they earn above or below the average salary.

**Solution:**

SELECT first\_name, last\_name, salary,

CASE

WHEN salary > (SELECT AVG(salary) FROM employees) THEN 'Above Average'

ELSE 'Below Average'

END AS salary\_category

FROM employees;

**18. Nested Subquery in WHERE Clause**

**Exercise:** Retrieve employees whose salary is higher than at least one employee in the IT department.

**Solution:**

SELECT first\_name, last\_name, salary

FROM employees

WHERE salary > ANY (SELECT salary FROM employees WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'IT'));

**19. Using Multiple Subqueries in SELECT**

**Exercise:** Retrieve employees along with their department name and average department salary.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name,

(SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id) AS avg\_dept\_salary

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;

**20. Using Subquery with UNION**

**Exercise:** Retrieve the highest and lowest salary in the company.

**Solution:**

SELECT salary FROM employees WHERE salary = (SELECT MAX(salary) FROM employees)

UNION

SELECT salary FROM employees WHERE salary = (SELECT MIN(salary) FROM employees);

**Database Schema:**

Let's first create the necessary tables and insert sample data.

CREATE DATABASE LabExercises;

USE LabExercises;

CREATE TABLE Students\_A (

student\_id INT PRIMARY KEY,

name VARCHAR(50),

course VARCHAR(50)

);

CREATE TABLE Students\_B (

student\_id INT PRIMARY KEY,

name VARCHAR(50),

course VARCHAR(50)

);

INSERT INTO Students\_A VALUES

(1, 'Alice', 'Computer Science'),

(2, 'Bob', 'Mathematics'),

(3, 'Charlie', 'Physics'),

(4, 'David', 'Computer Science');

INSERT INTO Students\_B VALUES

(3, 'Charlie', 'Physics'),

(4, 'David', 'Computer Science'),

(5, 'Eve', 'Biology'),

(6, 'Frank', 'Mathematics');

**Lab Exercises**

**Exercise 1: Use UNION to retrieve distinct students from both tables**

SELECT student\_id, name FROM Students\_A

UNION

SELECT student\_id, name FROM Students\_B;

**Exercise 2: Use UNION ALL to retrieve all students (including duplicates)**

SELECT student\_id, name FROM Students\_A

UNION ALL

SELECT student\_id, name FROM Students\_B;

**Exercise 3: Find students who are in both tables (INTERSECT equivalent)**

SELECT student\_id, name FROM Students\_A

INTERSECT

SELECT student\_id, name FROM Students\_B;

*MySQL does not support INTERSECT, so use this alternative:*

SELECT student\_id, name FROM Students\_A

WHERE student\_id IN (SELECT student\_id FROM Students\_B);

**Exercise 4: Find students who are in Students\_A but not in Students\_B (EXCEPT equivalent)**

SELECT student\_id, name FROM Students\_A

EXCEPT

SELECT student\_id, name FROM Students\_B;

*Alternative using NOT IN:*

SELECT student\_id, name FROM Students\_A

WHERE student\_id NOT IN (SELECT student\_id FROM Students\_B);

**Exercise 5: Find students who are in Students\_B but not in Students\_A**

SELECT student\_id, name FROM Students\_B

WHERE student\_id NOT IN (SELECT student\_id FROM Students\_A);

**Exercise 6: Use UNION with an ORDER BY clause**

SELECT student\_id, name FROM Students\_A

UNION

SELECT student\_id, name FROM Students\_B

ORDER BY name;

**Exercise 7: Use UNION ALL and count total students**

SELECT COUNT(\*) AS total\_students FROM (

SELECT student\_id FROM Students\_A

UNION ALL

SELECT student\_id FROM Students\_B

) AS combined\_students;

**Exercise 8: Find unique courses using UNION**

SELECT course FROM Students\_A

UNION

SELECT course FROM Students\_B;

**Exercise 9: Find students who are in only one of the tables**

SELECT student\_id, name FROM Students\_A

WHERE student\_id NOT IN (SELECT student\_id FROM Students\_B)

UNION

SELECT student\_id, name FROM Students\_B

WHERE student\_id NOT IN (SELECT student\_id FROM Students\_A);

**Exercise 10: Use UNION on a non-identical column list**

SELECT name, course FROM Students\_A

UNION

SELECT name, 'Unknown Course' FROM Students\_B;

**Exercise 11: Use UNION ALL with a WHERE condition**

SELECT student\_id, name FROM Students\_A WHERE course = 'Computer Science'

UNION ALL

SELECT student\_id, name FROM Students\_B WHERE course = 'Computer Science';

**Exercise 12: Use a subquery within UNION**

SELECT \* FROM (

SELECT student\_id, name FROM Students\_A

UNION

SELECT student\_id, name FROM Students\_B

) AS combined\_students

WHERE name LIKE 'C%';

**Exercise 13: Find students with common courses in both tables**

SELECT course FROM Students\_A

INTERSECT

SELECT course FROM Students\_B;

*Alternative:*

SELECT DISTINCT course FROM Students\_A

WHERE course IN (SELECT course FROM Students\_B);

**Exercise 14: Count distinct students across both tables**

SELECT COUNT(DISTINCT student\_id) FROM (

SELECT student\_id FROM Students\_A

UNION

SELECT student\_id FROM Students\_B

) AS combined\_students;

**Exercise 15: Find students who are in both tables but have different course names**

SELECT A.student\_id, A.name, A.course AS Course\_A, B.course AS Course\_B

FROM Students\_A A

JOIN Students\_B B ON A.student\_id = B.student\_id

WHERE A.course <> B.course;

**Exercise 16: Retrieve students from Students\_A but exclude those from Students\_B**

SELECT \* FROM Students\_A

WHERE student\_id NOT IN (SELECT student\_id FROM Students\_B);

**Exercise 17: Retrieve students from Students\_B but exclude those from Students\_A**

SELECT \* FROM Students\_B

WHERE student\_id NOT IN (SELECT student\_id FROM Students\_A);

**Exercise 18: UNION with hardcoded values**

SELECT 'From Students\_A' AS Source, name FROM Students\_A

UNION

SELECT 'From Students\_B', name FROM Students\_B;

**Exercise 19: UNION using a common table expression (CTE)**

WITH CombinedStudents AS (

SELECT student\_id, name FROM Students\_A

UNION

SELECT student\_id, name FROM Students\_B

)

SELECT \* FROM CombinedStudents;

**Exercise 20: Find students who exist in both tables with UNION and JOIN**

SELECT A.student\_id, A.name FROM Students\_A A

JOIN Students\_B B ON A.student\_id = B.student\_id

UNION

SELECT B.student\_id, B.name FROM Students\_B B

JOIN Students\_A A ON B.student\_id = A.student\_id;