**PL/SQL RDBMS Practical Interview Questions**

1. **Write a PL/SQL block to find the factorial of a number using a loop.**
   * **Task:** Create a PL/SQL block that takes an input number and calculates its factorial using a loop. Display the result.
2. **Create a procedure to insert a new employee record into the employees table.**
   * **Task:** Write a PL/SQL procedure named add\_employee that accepts parameters for employee details (e.g., name, position, salary) and inserts a new record into the employees table.
3. **Write a function to calculate the average salary of employees in a specific department.**
   * **Task:** Create a PL/SQL function named get\_average\_salary that takes a department ID as input and returns the average salary of employees in that department.
4. **Implement a trigger that automatically updates the last\_modified timestamp column whenever a record in the employees table is updated.**
   * **Task:** Write a PL/SQL trigger that fires before an update on the employees table and sets the last\_modified column to the current timestamp.
5. **Create a PL/SQL block that handles exceptions when trying to divide by zero.**
   * **Task:** Write a PL/SQL block that performs a division operation and includes exception handling to catch and display an error message if a division by zero occurs.
6. **Write a PL/SQL block to display all employees with a salary greater than a specified amount.**
   * **Task:** Create a PL/SQL block that prompts the user for a salary amount and then retrieves and displays the names and salaries of all employees earning more than that amount.
7. **Create a procedure that deletes employees who have not been active for more than a year.**
   * **Task:** Write a PL/SQL procedure named delete\_inactive\_employees that deletes records from the employees table where the last active date is more than one year ago.
8. **Implement a PL/SQL block that uses a cursor to fetch and display employee details.**
   * **Task:** Write a PL/SQL block that declares a cursor for the employees table, fetches each employee's details, and displays them in a loop.
9. **Create a function that checks if a given employee ID exists in the employees table.**
   * **Task:** Write a PL/SQL function named employee\_exists that takes an employee ID as input and returns TRUE if the ID exists, or FALSE otherwise.
10. **Write a PL/SQL block that uses a collection to store and display the names of all employees.**
    * **Task:** Create a PL/SQL block that declares a collection (e.g., an associative array) to hold employee names, populates it with names from the employees table, and then displays the names.

**SQL Queries (Practice it)**

1. **Select All Columns from a Table**

sql

SELECT \* FROM employees;

**Select Specific Columns**

sql

SELECT first\_name, last\_name, salary FROM employees;

**Filter Results with WHERE Clause**

sql

SELECT \* FROM employees WHERE department\_id = 10;

**Order Results**

sql

SELECT \* FROM employees ORDER BY salary DESC;

**Aggregate Functions**

sql

SELECT COUNT(\*) AS total\_employees FROM employees;

**Group By with Aggregate Function**

sql

SELECT department\_id, AVG(salary) AS average\_salary

FROM employees

GROUP BY department\_id;

**Join Two Tables**

sql

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

FROM employees e

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

**Subquery Example**

sql

SELECT first\_name, last\_name

FROM employees

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

**Update Records**

sql

UPDATE employees

SET salary = salary \* 1.10

WHERE department\_id = 20;

**Delete Records**

sql

DELETE FROM employees

1. WHERE hire\_date < '2020-01-01';

**PL/SQL Examples**

1. **Basic PL/SQL Block**

plsql

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hello, World!');

END;

**Variable Declaration and Output**

plsql

DECLARE

v\_employee\_name employees.first\_name%TYPE;

BEGIN

SELECT first\_name INTO v\_employee\_name FROM employees WHERE employee\_id = 1;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || v\_employee\_name);

END;

**Using IF Statement**

plsql

DECLARE

v\_salary employees.salary%TYPE;

BEGIN

SELECT salary INTO v\_salary FROM employees WHERE employee\_id = 1;

IF v\_salary > 5000 THEN

DBMS\_OUTPUT.PUT\_LINE('High Salary');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Normal Salary');

END IF;

END;

**Looping Through Records**

plsql

DECLARE

CURSOR emp\_cursor IS SELECT first\_name FROM employees;

v\_employee\_name emp\_cursor%ROWTYPE;

BEGIN

FOR v\_employee\_name IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_employee\_name.first\_name);

END LOOP;

END;

**Using a Procedure**

plsql

CREATE OR REPLACE PROCEDURE get\_employee\_name(p\_id IN NUMBER) IS

v\_name employees.first\_name%TYPE;

BEGIN

SELECT first\_name INTO v\_name FROM employees WHERE employee\_id = p\_id;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || v\_name);

END;

**Using a Function**

plsql

CREATE OR REPLACE FUNCTION get\_employee\_salary(p\_id IN NUMBER) RETURN NUMBER IS

v\_salary employees.salary%TYPE;

BEGIN

SELECT salary INTO v\_salary FROM employees WHERE employee\_id = p\_id;

RETURN v\_salary;

END;

**Exception Handling**

plsql

DECLARE

v\_salary employees.salary%TYPE;

BEGIN

SELECT salary INTO v\_salary FROM employees WHERE employee\_id = 999; -- Assuming this ID does not exist

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee not found.');

END;

**Using a Trigger**

plsql

CREATE OR REPLACE TRIGGER before\_insert\_employee

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

:new.hire\_date := SYSDATE;

END;

**Using Collections**

plsql

DECLARE

TYPE emp\_table IS TABLE OF employees%ROWTYPE;

v\_employees emp\_table;

BEGIN

SELECT \* BULK COLLECT INTO v\_employees FROM employees;

FOR i IN 1..v\_employees.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_employees(i).first\_name);

END LOOP;

END;

**Using Cursors with Parameters**

plsql

DECLARE

CURSOR emp\_cursor(p\_dept\_id IN NUMBER) IS

SELECT first\_name FROM employees WHERE department\_id = p\_dept\_id;

v\_employee\_name

**SQL and PL/SQL query scenario based advance practicals**

**Advanced SQL Queries**

1. **Find Employees with the Highest Salary in Each Department**

sql

SELECT department\_id, first\_name, last\_name, salary

FROM employees e1

WHERE salary = (

SELECT MAX(salary)

FROM employees e2

WHERE e1.department\_id = e2.department\_id

);

**Rank Employees by Salary within Each Department**

sql

SELECT first\_name, last\_name, department\_id, salary,

RANK() OVER (PARTITION BY department\_id ORDER BY salary DESC) AS salary\_rank

FROM employees;

**Find Departments with More than 5 Employees**

sql

SELECT department\_id, COUNT(\*) AS employee\_count

FROM employees

GROUP BY department\_id

HAVING COUNT(\*) > 5;

**Find Employees Who Joined in the Last 30 Days**

sql

SELECT first\_name, last\_name, hire\_date

FROM employees

WHERE hire\_date >= SYSDATE - 30;

**Calculate the Total Salary Expense for Each Department**

sql

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

FROM employees

GROUP BY department\_id;

**Find Employees with Salaries Above the Average Salary of Their Department**

sql

SELECT first\_name, last\_name, salary

FROM employees e1

WHERE salary > (

SELECT AVG(salary)

FROM employees e2

WHERE e1.department\_id = e2.department\_id

);

**Create a View for Employees with High Salaries**

sql

CREATE VIEW high\_salary\_employees AS

SELECT first\_name, last\_name, salary

FROM employees

WHERE salary > 100000;

**Use Common Table Expressions (CTE) to Find the Top 3 Highest Paid Employees**

sql

WITH ranked\_employees AS (

SELECT first\_name, last\_name, salary,

RANK() OVER (ORDER BY salary DESC) AS salary\_rank

FROM employees

)

SELECT first\_name, last\_name, salary

FROM ranked\_employees

WHERE salary\_rank <= 3;

**Find Employees Who Have Not Been Assigned to Any Project**

sql

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

FROM employees e

LEFT JOIN project\_assignments pa ON e.employee\_id = pa.employee\_id

WHERE pa.project\_id IS NULL;

**Generate a Report of Employee Count by Hire Year**

sql

SELECT EXTRACT(YEAR FROM hire\_date) AS hire\_year, COUNT(\*) AS employee\_count

FROM employees

GROUP BY EXTRACT(YEAR FROM hire\_date)

ORDER BY hire\_year;

**Advanced PL/SQL Scenarios**

1. **Bulk Insert from One Table to Another**

plsql

DECLARE

TYPE emp\_rec IS RECORD (

first\_name employees.first\_name%TYPE,

last\_name employees.last\_name%TYPE

);

TYPE emp\_tab IS TABLE OF emp\_rec;

v\_employees emp\_tab;

BEGIN

SELECT first\_name, last\_name BULK COLLECT INTO v\_employees FROM employees WHERE department\_id = 10;

FORALL i IN v\_employees.FIRST .. v\_employees.LAST

INSERT INTO high\_salary\_employees (first\_name, last\_name) VALUES (v\_employees(i).first\_name, v\_employees(i).last\_name);

END;

**Creating a Function to Calculate Bonus**

plsql

CREATE OR REPLACE FUNCTION calculate\_bonus(p\_salary IN NUMBER) RETURN NUMBER IS

BEGIN

RETURN p\_salary \* 0.10; -- 10% bonus

END;

**Using a Cursor to Update Employee Salaries**

plsql

DECLARE

CURSOR emp\_cursor IS SELECT employee\_id, salary FROM employees WHERE department\_id = 20;

v\_salary employees.salary%TYPE;

BEGIN

FOR emp\_record IN emp\_cursor LOOP

v\_salary := emp\_record.salary \* 1.05; -- Increase salary by 5%

UPDATE employees SET salary = v\_salary WHERE employee\_id = emp\_record.employee\_id;

END LOOP;

END;

**Creating a Trigger to Log Changes to Employee Salaries**

plsql

CREATE OR REPLACE TRIGGER salary\_change\_log

AFTER UPDATE OF salary ON employees

FOR EACH ROW

BEGIN

INSERT INTO salary\_history (employee\_id, old\_salary, new\_salary, change\_date)

VALUES (:OLD.employee\_id, :OLD.salary, :NEW.salary, SYSDATE );

END;

**Using a Package to Group Related Procedures and Functions**

plsql

CREATE OR REPLACE PACKAGE employee\_management AS

PROCEDURE add\_employee(p\_first\_name IN VARCHAR2, p\_last\_name IN VARCHAR2, p\_salary IN NUMBER);

FUNCTION get\_employee\_count RETURN NUMBER;

END employee\_management;

**Implementing a Procedure to Delete Employees Based on Criteria**

plsql

CREATE OR REPLACE PROCEDURE delete\_old\_employees(p\_years IN NUMBER) IS

BEGIN

DELETE FROM employees WHERE hire\_date < ADD\_MONTHS(SYSDATE, -12 \* p\_years);

END;

**Using Dynamic SQL to Execute Queries**

plsql

DECLARE

v\_sql VARCHAR2(1000);

v\_count NUMBER;

BEGIN

v\_sql := 'SELECT COUNT(\*) FROM employees WHERE department\_id = :dept\_id';

EXECUTE IMMEDIATE v\_sql INTO v\_count USING 10;

DBMS\_OUTPUT.PUT\_LINE('Employee Count: ' || v\_count);

END;

**Creating a Recursive Function to Calculate Factorial**

plsql

CREATE OR REPLACE FUNCTION factorial(n IN NUMBER) RETURN NUMBER IS

BEGIN

IF n = 0 THEN

RETURN 1;

ELSE

RETURN n \* factorial(n - 1);

END IF;

END;

**Using a Cursor to Fetch and Process Data**

plsql

DECLARE

CURSOR emp\_cursor IS SELECT employee\_id, first\_name FROM employees;

v\_employee\_id employees.employee\_id%TYPE;

v\_first\_name employees.first\_name%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO v\_employee\_id, v\_first\_name;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || v\_employee\_id || ', Name: ' || v\_first\_name);

END LOOP;

CLOSE emp\_cursor;

END;

**Creating a Procedure to Send Notifications**

plsql

CREATE OR REPLACE PROCEDURE send\_notification(p\_employee\_id IN NUMBER, p\_message IN VARCHAR2) IS

BEGIN

-- Logic to send notification (e.g., email, message)

DBMS\_OUTPUT.PUT\_LINE('Notification sent to Employee ID: ' || p\_employee\_id || ' with message: ' || p\_message);

END;

``` ### Advanced SQL Queries (continued)

**Find Employees with the Same Last Name**

sql

SELECT last\_name, COUNT(\*) AS name\_count

FROM employees

GROUP BY last\_name

HAVING COUNT(\*) > 1;

**Calculate Yearly Salary for Each Employee**

sql

SELECT first\_name, last\_name, salary \* 12 AS yearly\_salary

FROM employees;

**Identify Employees with Missing Email Addresses**

sql

SELECT first\_name, last\_name

FROM employees

WHERE email IS NULL OR email = '';

**Find the Second Highest Salary**

sql

SELECT MAX(salary) AS second\_highest\_salary

FROM employees

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

**List Employees and Their Managers**

sql

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

FROM employees e

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

**Calculate the Percentage of Employees in Each Department**

sql

SELECT department\_id, COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM employees) AS percentage

FROM employees

GROUP BY department\_id;

**Find Employees Who Have Worked on Multiple Projects**

sql

SELECT e.first\_name, e.last\_name, COUNT(pa.project\_id) AS project\_count

FROM employees e

JOIN project\_assignments pa ON e.employee\_id = pa.employee\_id

GROUP BY e.first\_name, e.last\_name

HAVING COUNT(pa.project\_id) > 1;

**Get the Total Number of Employees Hired Each Month**

sql

SELECT TO\_CHAR(hire\_date, 'YYYY-MM') AS hire\_month, COUNT(\*) AS employee\_count

FROM employees

GROUP BY TO\_CHAR(hire\_date, 'YYYY-MM')

ORDER BY hire\_month;

**Find Employees with Salaries in the Top 10%**

sql

SELECT first\_name, last\_name, salary

FROM employees

WHERE salary > (SELECT PERCENTILE\_CONT(0.90) WITHIN GROUP (ORDER BY salary) FROM employees);

**Create a Temporary Table for Employee Data**

sql

CREATE GLOBAL TEMPORARY TABLE temp\_employees AS

SELECT \* FROM employees WHERE ROWNUM <= 10;

1. **Creating a Procedure to Update Employee Information**

plsql

CREATE OR REPLACE PROCEDURE update\_employee\_info(p\_id IN NUMBER, p\_salary IN NUMBER, p\_department\_id IN NUMBER) IS

BEGIN

UPDATE employees

SET salary = p\_salary, department\_id = p\_department\_id

WHERE employee\_id = p\_id;

END;

**Using a Function to Get Employee Full Name**

plsql

CREATE OR REPLACE FUNCTION get\_full\_name(p\_id IN NUMBER) RETURN VARCHAR2 IS

v\_full\_name VARCHAR2(100);

BEGIN

SELECT first\_name || ' ' || last\_name INTO v\_full\_name FROM employees WHERE employee\_id = p\_id;

RETURN v\_full\_name;

END;

**Creating a Trigger to Prevent Salary Below Minimum**

plsql

CREATE OR REPLACE TRIGGER check\_salary\_before\_insert

BEFORE INSERT OR UPDATE ON employees

FOR EACH ROW

BEGIN

IF :NEW.salary < 3000 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Salary cannot be less than 3000');

END IF;

END;

**Using a Cursor to Generate a Report**

plsql

DECLARE

CURSOR emp\_cursor IS SELECT first\_name, last\_name, salary FROM employees;

BEGIN

FOR emp\_record IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE(emp\_record.first\_name || ' ' || emp\_record.last\_name || ': ' || emp\_record.salary);

END LOOP;

END;

**Creating a Package for Employee Management**

plsql

CREATE OR REPLACE PACKAGE employee\_pkg AS

PROCEDURE hire\_employee(p\_first\_name IN VARCHAR2, p\_last\_name IN VARCHAR2, p\_salary IN NUMBER);

FUNCTION get\_employee\_salary(p\_id IN NUMBER) RETURN NUMBER;

END employee\_pkg;

**Implementing a Function to Calculate Total Compensation**

plsql

CREATE OR REPLACE FUNCTION calculate\_total\_compensation(p\_id IN NUMBER) RETURN NUMBER IS

v\_salary employees.salary%TYPE;

v\_bonus NUMBER := 0.10; -- 10% bonus

BEGIN

SELECT salary INTO v\_salary FROM employees WHERE employee\_id = p\_id;

RETURN v\_salary + (v\_salary \* v\_bonus);

END;

**Creating a Procedure to Archive Old Employees**

plsql

CREATE OR REPLACE PROCEDURE archive\_old\_employees(p\_years IN NUMBER) IS

BEGIN

INSERT INTO archived\_employees (employee\_id, first\_name, last\_name, hire\_date)

SELECT employee\_id, first\_name, last\_name, hire\_date

FROM employees

WHERE hire\_date < ADD\_MONTHS(SYSDATE, -12 \* p\_years);

DELETE FROM employees WHERE hire\_date < ADD\_MONTHS(SYSDATE, -12 \* p\_years);

END;

**Using a Function to Check Employee Existence**

plsql

CREATE OR REPLACE FUNCTION employee\_exists(p\_id IN NUMBER) RETURN BOOLEAN IS

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM employees WHERE employee\_id = p\_id;

RETURN v\_count > 0;

END;

**Creating a Trigger to Log Employee Deletions**

plsql

CREATE OR REPLACE TRIGGER log\_employee\_deletion

AFTER DELETE ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_deletion\_log (employee\_id, deletion\_date)

VALUES (:OLD.employee\_id, SYSDATE);

END;

**Using a Cursor to Calculate Average Salary by Department**

plsql

DECLARE

CURSOR dept\_cursor IS SELECT department\_id FROM departments;

v\_avg\_salary NUMBER;

BEGIN

FOR dept\_record IN dept\_cursor LOOP

SELECT AVG(salary) INTO v\_avg\_salary FROM employees WHERE department\_id = dept\_record.department\_id;

DBMS\_OUTPUT.PUT\_LINE('Department ID: ' || dept\_record.department\_id || ', Average Salary: ' || v\_avg\_salary);

END LOOP;

END;

**SQL Query Optimization Example**

SQL query optimization is the process of improving the performance of a SQL query by reducing its execution time and resource consumption. Here’s an example of a poorly performing SQL query and how to optimize it.

**Example Scenario**

Suppose we have the following tables:

1. **employees**: Contains employee details.
   * employee\_id (Primary Key)
   * first\_name
   * last\_name
   * department\_id
   * salary
   * hire\_date
2. **departments**: Contains department details.
   * department\_id (Primary Key)
   * department\_name

**Poorly Performing Query**

Here’s a query that retrieves the names of employees who earn more than the average salary of their department:

sql

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 e.salary > (

SELECT AVG(salary)

FROM employees

WHERE department\_id = e.department\_id

);

**Issues with the Query**

1. **Correlated Subquery**: The subquery that calculates the average salary is executed for each employee, which can lead to poor performance, especially with a large dataset.
2. **Lack of Indexes**: If there are no indexes on department\_id in the employees table or on salary, the query will perform full table scans.

**Optimized Query**

To optimize the query, we can use a Common Table Expression (CTE) or a derived table to calculate the average salary for each department once, and then join it with the employees table.

**Using a CTE**

sql

WITH avg\_salaries AS (

SELECT department\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department\_id

)

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

JOIN avg\_salaries a ON e.department\_id = a.department\_id

WHERE e.salary > a.avg\_salary;

**Using a Derived Table**

sql

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

JOIN (

SELECT department\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department\_id

) a ON e.department\_id = a.department\_id

WHERE e.salary > a.avg\_salary;

**Benefits of Optimization**

1. **Reduced Execution Time**: The average salary is calculated once instead of multiple times, significantly reducing the execution time.
2. **Improved Readability**: The use of CTEs or derived tables makes the query easier to read and understand.
3. **Better Resource Utilization**: By reducing the number of calculations and potential full table scans, the query uses fewer resources.

**Additional Optimization Techniques**

1. **Indexing**: Ensure that there are indexes on columns used in joins and where clauses, such as department\_id and salary.
2. **Analyze Execution Plans**: Use tools like EXPLAIN to analyze the execution plan of the query and identify bottlenecks.
3. **Limit Result Set**: If applicable, use LIMIT or TOP to restrict the number of rows returned.
4. \*\*Avoid SELECT \*\*\*: Only select the columns you need instead of using SELECT \*.
5. **Partitioning**: For very large tables, consider partitioning to improve query performance.

**Examples of Joint Query with multiple tables**

Below examples of SQL queries involve joining multiple tables. These examples will demonstrate different types of joins (INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN) and how to work with multiple tables in a relational database.

**Example Tables**

Let's assume we have the following tables:

1. **employees**
   * employee\_id (Primary Key)
   * first\_name
   * last\_name
   * department\_id
   * salary
2. **departments**
   * department\_id (Primary Key)
   * department\_name
3. **projects**
   * project\_id (Primary Key)
   * project\_name
   * department\_id
4. **project\_assignments**
   * assignment\_id (Primary Key)
   * employee\_id
   * project\_id

**1. INNER JOIN**

**Query**: Retrieve a list of employees along with their department names.

sql

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. LEFT JOIN**

**Query**: Retrieve a list of all employees and their department names, including employees who are not assigned to any department.

sql

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

FROM employees e

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

**3. RIGHT JOIN**

**Query**: Retrieve a list of all departments and the employees assigned to them, including departments that have no employees.

sql

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

FROM departments d

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

**4. FULL OUTER JOIN**

**Query**: Retrieve a list of all employees and departments, showing employees without departments and departments without employees.

sql

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

FROM employees e

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

**5. Joining Three Tables**

**Query**: Retrieve a list of employees, their department names, and projects they are assigned to.

sql

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

FROM employees e

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

INNER JOIN project\_assignments pa ON e.employee\_id = pa.employee\_id

INNER JOIN projects p ON pa.project\_id = p.project\_id;

**6. Using Aggregate Functions with Joins**

**Query**: Retrieve the total salary of employees in each department.

sql

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;

**7. Joining with Conditions**

**Query**: Retrieve a list of employees who are assigned to projects in a specific department.

sql

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

FROM employees e

INNER JOIN project\_assignments pa ON e.employee\_id = pa.employee\_id

INNER JOIN projects p ON pa.project\_id = p.project\_id

WHERE p.department\_id = 1; -- Assuming department\_id 1 is the target department

**8. Self Join**

**Query**: Retrieve a list of employees and their managers (assuming managers are also employees).

sql

SELECT e1.first\_name AS employee\_name, e2.first\_name AS manager\_name

FROM employees e1

LEFT JOIN employees e2 ON e1.manager\_id = e2.employee\_id;

**9. Joining with Subqueries**

**Query**: Retrieve employees who earn more than the average salary of their department.

sql

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

FROM employees e

WHERE e.salary > (

SELECT AVG(salary)

FROM employees

WHERE department\_id = e.department\_id

);

**10. Complex Join with Multiple Conditions**

**Query**: Retrieve a list of employees, their departments, and projects, filtering by specific criteria.

sql

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

FROM employees e

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

INNER JOIN project\_assignments pa ON e.employee\_id = pa.employee\_id

INNER JOIN projects p ON pa.project\_id = p.project\_id

WHERE d.department\_name = 'Sales' AND e.salary > 50000;

**Summary**

These examples illustrate how to perform various types of joins with multiple tables in SQL. Depending on your specific requirements, you can adjust the queries to fit your data model and the information you need to retrieve.