1A) Queries (along with sub-queries) using ANY, ALL, IN, EXISTS, NOT EXISTS.

UNION, INTERSECT, Constraints.

-- Employees Table  
CREATE TABLE Employees (  
    emp\_id NUMBER PRIMARY KEY,  
    name VARCHAR2(50) NOT NULL,  
    department\_id NUMBER,  
    salary NUMBER CHECK (salary > 0),  
    FOREIGN KEY (department\_id) REFERENCES Departments(department\_id)  
);  
  
-- ProjectAssignments Table  
CREATE TABLE ProjectAssignments (  
    emp\_id NUMBER,  
    project\_name VARCHAR2(50),  
    FOREIGN KEY (emp\_id) REFERENCES Employees(emp\_id)  
);  
INSERT INTO Departments (department\_id, dept\_name) VALUES (10, 'Engineering');  
INSERT INTO Departments (department\_id, dept\_name) VALUES (20, 'Marketing');  
INSERT INTO Departments (department\_id, dept\_name) VALUES (30, 'HR');  
INSERT INTO Departments (department\_id, dept\_name) VALUES (40, 'Finance');  
INSERT INTO Employees (emp\_id, name, department\_id, salary) VALUES (1, 'Alice', 10, 50000);  
INSERT INTO Employees (emp\_id, name, department\_id, salary) VALUES (2, 'Bob', 20, 60000);  
INSERT INTO Employees (emp\_id, name, department\_id, salary) VALUES (3, 'Charlie', 10, 55000);  
INSERT INTO Employees (emp\_id, name, department\_id, salary) VALUES (4, 'Dave', 30, 70000);  
INSERT INTO Employees (emp\_id, name, department\_id, salary) VALUES (5, 'Eve', 20, 40000);  
INSERT INTO ProjectAssignments (emp\_id, project\_name) VALUES (1, 'Project X');  
INSERT INTO ProjectAssignments (emp\_id, project\_name) VALUES (2, 'Project Y');  
INSERT INTO ProjectAssignments (emp\_id, project\_name) VALUES (4, 'Project X');  
INSERT INTO ProjectAssignments (emp\_id, project\_name) VALUES (5, 'Project Z');  
--Any  
SELECT name, salary  
FROM Employees  
WHERE salary > ANY (  
    SELECT salary  
    FROM Employees  
    WHERE department\_id = 20  
);  
--All  
SELECT name, salary  
FROM Employees  
WHERE salary > ALL (  
    SELECT salary FROM Employees WHERE department\_id = 20  
);  
--Exists  
SELECT dept\_name  
FROM Departments d  
WHERE EXISTS (  
    SELECT 1 FROM Employees e  
    WHERE e.department\_id = d.department\_id  
);  
--Not exists  
SELECT dept\_name  
FROM Departments d  
WHERE NOT EXISTS (  
    SELECT 1 FROM Employees e  
    WHERE e.department\_id = d.department\_id  
);  
--union  
SELECT name FROM Employees WHERE department\_id = 10  
UNION  
SELECT name FROM Employees WHERE department\_id = 30;  
--intersect  
SELECT name FROM Employees WHERE department\_id = 20  
INTERSECT  
SELECT name FROM Employees WHERE salary < 60000;

B) Develop programs using features parameters in a CURSOR variable.

-Cursors  
-- Enable DBMS\_OUTPUT in SQL Developer or SQLcl  
SET SERVEROUTPUT ON;  
DECLARE  
    -- Cursor with IN parameter  
    CURSOR emp\_cursor(dept\_id IN Employees.department\_id%TYPE) IS  
        SELECT emp\_id, name, salary  
        FROM Employees  
        WHERE department\_id = dept\_id;  
    -- Variables to hold values from cursor  
    v\_emp\_id Employees.emp\_id%TYPE;  
    v\_name Employees.name%TYPE;  
    v\_salary Employees.salary%TYPE;  
BEGIN  
    -- Call the cursor with department\_id = 10 (Engineering)  
    OPEN emp\_cursor(10);  
    LOOP  
        FETCH emp\_cursor INTO v\_emp\_id, v\_name, v\_salary;  
        EXIT WHEN emp\_cursor%NOTFOUND;  
        -- Print the values  
        DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_emp\_id || ', Name: ' || v\_name || ', Salary: ' || v\_salary);  
    END LOOP;  
    CLOSE emp\_cursor;  
END;  
/

2A) Queries (along with sub-queries) using UNION, INTERSECT Constraints.

--union  
SELECT name FROM Employees WHERE department\_id = 10  
UNION  
SELECT name FROM Employees WHERE department\_id = 30;  
--intersect  
SELECT name FROM Employees WHERE department\_id = 20  
INTERSECT  
SELECT name FROM Employees WHERE salary < 60000;

B) Program development using WHILE LOOPS, numeric FOR LOOPS using ERROR Handling, BUILT –IN Exceptions, USER defined Exceptions, RAISE-APPLICATION ERROR.

SET SERVEROUTPUT ON;  
DECLARE  
    -- Cursor with parameter and FOR UPDATE  
    CURSOR emp\_cursor(p\_dept\_id IN NUMBER) IS  
        SELECT emp\_id, name, salary  
        FROM Employees  
        WHERE department\_id = p\_dept\_id  
        FOR UPDATE;  
    -- Variables to hold data  
    v\_emp\_id Employees.emp\_id%TYPE;  
    v\_name   Employees.name%TYPE;  
    v\_salary Employees.salary%TYPE;  
    v\_new\_salary Employees.salary%TYPE;  
    -- User-defined exception  
    e\_low\_salary EXCEPTION;  
BEGIN  
    -- Open cursor for Department 10  
    OPEN emp\_cursor(10);  
    LOOP  
        FETCH emp\_cursor INTO v\_emp\_id, v\_name, v\_salary;  
        EXIT WHEN emp\_cursor%NOTFOUND;  
        -- Raise user-defined exception if salary is too low  
        IF v\_salary < 10000 THEN  
            RAISE e\_low\_salary;  
        END IF;  
        -- Raise built-in exception if name is NULL  
        IF v\_name IS NULL THEN  
            RAISE\_APPLICATION\_ERROR(-20001, 'Employee name cannot be NULL (ID: ' || v\_emp\_id || ')');  
        END IF;  
        -- Calculate new salary (increase by 10%)  
        v\_new\_salary := v\_salary \* 1.10;  
        -- Update using WHERE CURRENT OF  
        UPDATE Employees  
        SET salary = v\_new\_salary  
        WHERE CURRENT OF emp\_cursor;  
        -- Output info  
        DBMS\_OUTPUT.PUT\_LINE('Updated Salary for ' || v\_name || ' (ID: ' || v\_emp\_id || ') to: ' || v\_new\_salary);  
    END LOOP;  
    COMMIT;  
    CLOSE emp\_cursor;  
-- Exception handling  
EXCEPTION  
    WHEN e\_low\_salary THEN  
        DBMS\_OUTPUT.PUT\_LINE(' Salary is too low for update.');  
    WHEN OTHERS THEN  
        DBMS\_OUTPUT.PUT\_LINE(' Error: ' || SQLERRM);  
END;  
/

3A)Queries using Aggregate functions (COUNT, SUM, AVG, MAX and MIN), GROUPBY, HAVING and Creation and dropping of Views.

--Count  
SELECT COUNT(\*) AS total\_employees FROM Employees;  
--Sum  
SELECT SUM(salary) AS total\_salary FROM Employees;  
--Average value  
SELECT AVG(salary) AS avg\_salary FROM Employees;  
--Max and min  
SELECT MAX(salary) AS highest\_salary, MIN(salary) AS lowest\_salary FROM Employees;  
--Group by  
SELECT department\_id, COUNT(\*) AS emp\_count, AVG(salary) AS avg\_salary  
FROM Employees  
GROUP BY department\_id;  
--Having  
SELECT department\_id, AVG(salary) AS avg\_salary  
FROM Employees  
GROUP BY department\_id  
HAVING AVG(salary) > 50000;  
--Creating a view  
CREATE VIEW dept\_summary AS  
SELECT department\_id, COUNT(\*) AS emp\_count, AVG(salary) AS avg\_salary  
FROM Employees  
GROUP BY department\_id;  
--query  
SELECT \* FROM dept\_summary;  
--dropping view  
DROP VIEW dept\_summary;

|  |
| --- |
|  |

B)Implement a function which accepts employee id as input and has to return the job id of the employee.

ALTER TABLE Employees ADD job\_id VARCHAR2(20);  
UPDATE Employees SET job\_id = 'DEV' WHERE emp\_id = 1;  
UPDATE Employees SET job\_id = 'HR' WHERE emp\_id = 2;  
UPDATE Employees SET job\_id = 'MKT' WHERE emp\_id = 3;  
UPDATE Employees SET job\_id = 'MKT' WHERE emp\_id = 4;  
UPDATE Employees SET job\_id = 'DEV' WHERE emp\_id = 5;  
COMMIT;  
select \* from employees  
--function for getting jobid and input is empid  
CREATE OR REPLACE FUNCTION get\_job\_id(p\_emp\_id IN NUMBER)  
RETURN VARCHAR2  
IS  
    v\_job\_id VARCHAR2(20);  -- To hold job\_id  
BEGIN  
    -- Fetch job\_id of the given employee  
    SELECT job\_id INTO v\_job\_id  
    FROM Employees  
    WHERE emp\_id = p\_emp\_id;  
  
    RETURN v\_job\_id;  -- Return the job\_id  
  
EXCEPTION  
    WHEN NO\_DATA\_FOUND THEN  
        RETURN 'Not Found';  -- If no employee with given ID  
    WHEN OTHERS THEN  
        RETURN 'Error';      -- For any other error  
END;  
/  
-- See result in a query  
SELECT get\_job\_id(1) FROM dual;  
  
-- Or in a PL/SQL block:  
BEGIN  
    DBMS\_OUTPUT.PUT\_LINE('Job ID: ' || get\_job\_id(1));  
END;  
/

|  |  |
| --- | --- |
|  |  |

4A)Queries using GROUPBY, HAVING.

--Group by  
SELECT department\_id, COUNT(\*) AS emp\_count, AVG(salary) AS avg\_salary  
FROM Employees  
GROUP BY department\_id;  
--Having  
SELECT department\_id, AVG(salary) AS avg\_salary  
FROM Employees  
GROUP BY department\_id  
HAVING AVG(salary) > 50000;

B) Programs development using creation of procedures, passing parameters IN and OUT of PROCEDURES.

CREATE OR REPLACE PROCEDURE get\_employee\_job (  
    p\_emp\_id   IN  Employees.emp\_id%TYPE,  
    p\_job\_id   OUT Employees.job\_id%TYPE  
)  
IS  
BEGIN  
    SELECT job\_id INTO p\_job\_id  
    FROM Employees  
    WHERE emp\_id = p\_emp\_id;  
  
EXCEPTION  
    WHEN NO\_DATA\_FOUND THEN  
        p\_job\_id := 'Not Found';  
    WHEN OTHERS THEN  
        p\_job\_id := 'Error';  
END;  
/  
DECLARE  
    v\_job\_id VARCHAR2(20);  -- OUT variable to hold job ID  
BEGIN  
    get\_employee\_job(1, v\_job\_id);  -- Call the procedure with emp\_id = 1  
    DBMS\_OUTPUT.PUT\_LINE('Job ID is: ' || v\_job\_id);  
END;  
/

1. A)Queries for Creation and dropping of Views.

--Creating a view  
CREATE VIEW dept\_summary AS  
SELECT department\_id, COUNT(\*) AS emp\_count, AVG(salary) AS avg\_salary  
FROM Employees  
GROUP BY department\_id;  
--query  
SELECT \* FROM dept\_summary;  
--dropping view  
DROP VIEW dept\_summary;

B) Develop a program that includes the features NESTEDIF, CASE and CASE expression. The program can be extended using the NULLIF and COALESCE functions.

SET SERVEROUTPUT ON;

DECLARE

-- Employee data variables

v\_emp\_id Employees.emp\_id%TYPE := 1; -- Test with emp\_id = 1

v\_name Employees.name%TYPE;

v\_salary Employees.salary%TYPE;

v\_job\_id Employees.job\_id%TYPE;

-- Output/logic variables

v\_bonus NUMBER;

v\_category VARCHAR2(30);

v\_null\_check VARCHAR2(50);

v\_final\_job\_id VARCHAR2(20);

BEGIN

-- Get employee data

SELECT name, salary, job\_id INTO v\_name, v\_salary, v\_job\_id

FROM Employees

WHERE emp\_id = v\_emp\_id;

-----------------------------------------

-- NESTED IF: Check salary ranges

-----------------------------------------

IF v\_salary > 60000 THEN

IF v\_salary > 80000 THEN

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

ELSE

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

END IF;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Salary: Average or Low');

END IF;

-----------------------------------------

-- CASE Statement: Determine job category

-----------------------------------------

CASE v\_job\_id

WHEN 'DEV' THEN v\_category := 'Developer';

WHEN 'HR' THEN v\_category := 'Human Resources';

WHEN 'MKT' THEN v\_category := 'Marketing';

ELSE v\_category := 'Other';

END CASE;

DBMS\_OUTPUT.PUT\_LINE('Job Category: ' || v\_category);

-----------------------------------------

-- CASE Expression: Calculate bonus

-----------------------------------------

v\_bonus := CASE

WHEN v\_salary > 80000 THEN 5000

WHEN v\_salary > 60000 THEN 3000

ELSE 1000

END;

DBMS\_OUTPUT.PUT\_LINE('Bonus: ' || v\_bonus);

-----------------------------------------

-- NULLIF: Check if salary is exactly 50000

-----------------------------------------

IF NULLIF(v\_salary, 50000) IS NULL THEN

v\_null\_check := 'Salary is exactly 50000';

ELSE

v\_null\_check := 'Salary is not 50000';

END IF;

DBMS\_OUTPUT.PUT\_LINE(v\_null\_check);

-----------------------------------------

-- COALESCE: Handle NULL job\_id

-----------------------------------------

v\_final\_job\_id := COALESCE(v\_job\_id, 'UNASSIGNED');

DBMS\_OUTPUT.PUT\_LINE('Final Job ID: ' || v\_final\_job\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

6 A)Queries using Conversion functions (to\_char, to\_number and to\_date).

--To char

SELECT name, TO\_CHAR(salary, '999,999') AS formatted\_salary

FROM Employees;

--SELECT name, TO\_CHAR(hire\_date, 'DD-Mon-YYYY') AS formatted\_date

FROM Employees;

--To number

SELECT TO\_NUMBER('10000') + 5000 AS total

FROM dual;

--To date

SELECT TO\_DATE('25-12-2023', 'DD-MM-YYYY') AS xmas

FROM dual;

--To date

SELECT TO\_CHAR(TO\_DATE('2025-03-15', 'YYYY-MM-DD'), 'Month DD, YYYY') AS formatted

FROM dual;

B) Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section.

SET SERVEROUTPUT ON;

DECLARE

-- DECLARATION SECTION

v\_emp\_id Employees.emp\_id%TYPE := 1; -- Input (change as needed)

v\_name Employees.name%TYPE; -- To store name

BEGIN

-- ️ EXECUTABLE SECTION

SELECT name INTO v\_name

FROM Employees

WHERE emp\_id = v\_emp\_id;

-- Print the employee name

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

EXCEPTION

-- ❗ EXCEPTION-HANDLING SECTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with ID: ' || v\_emp\_id);

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

/

1. A)Queries using string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr).

--Concatenation

SELECT name || ' works as ' || job\_id AS employee\_info

FROM Employees;

--LPAD

SELECT LPAD(name, 10, '\*') AS padded\_name

FROM Employees;

--RPAD

SELECT RPAD(name, 10, '\*') AS padded\_name

FROM Employees;

--LTRIM

SELECT LTRIM('\*\*\*Hello', '\*') AS trimmed

FROM dual;

--RTRIM

SELECT RTRIM('Hello\*\*\*', '\*') AS trimmed

FROM dual;

--LOWER

SELECT LOWER(name) AS lowercase\_name

FROM Employees;

--UPPER

SELECT UPPER(name) AS uppercase\_name

FROM Employees;

--INITCAP

SELECT INITCAP(name) AS proper\_case\_name

FROM Employees;

--LENGTH

SELECT name, LENGTH(name) AS name\_length

FROM Employees;

--SUBSTR

SELECT name, SUBSTR(name, 1, 3) AS first\_3\_letters

FROM Employees;

--INSTR

SELECT name, INSTR(name, 'a') AS position\_of\_a

FROM Employees;

--BONUS

SELECT INITCAP(LPAD(name, 10, '\*')) AS formatted\_name

FROM Employees;

B) Develop programs using features parameters in a CURSOR variables.

-Cursors  
-- Enable DBMS\_OUTPUT in SQL Developer or SQLcl  
SET SERVEROUTPUT ON;  
DECLARE  
    -- Cursor with IN parameter  
    CURSOR emp\_cursor(dept\_id IN Employees.department\_id%TYPE) IS  
        SELECT emp\_id, name, salary  
        FROM Employees  
        WHERE department\_id = dept\_id;  
    -- Variables to hold values from cursor  
    v\_emp\_id Employees.emp\_id%TYPE;  
    v\_name Employees.name%TYPE;  
    v\_salary Employees.salary%TYPE;  
BEGIN  
    -- Call the cursor with department\_id = 10 (Engineering)  
    OPEN emp\_cursor(10);  
    LOOP  
        FETCH emp\_cursor INTO v\_emp\_id, v\_name, v\_salary;  
        EXIT WHEN emp\_cursor%NOTFOUND;  
        -- Print the values  
        DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_emp\_id || ', Name: ' || v\_name || ', Salary: ' || v\_salary);  
    END LOOP;  
    CLOSE emp\_cursor;  
END;  
/

8A) Queries using date functions (Sysdate, next\_day, add\_months, last\_day, months\_between, least, greatest, trunc, round, to\_char, to\_date).

--sys date

SELECT SYSDATE AS current\_date FROM dual;

--Next day

SELECT NEXT\_DAY(SYSDATE, 'MONDAY') AS next\_monday FROM dual;

--add months

SELECT ADD\_MONTHS(SYSDATE, 3) AS date\_after\_3\_months FROM dual;

--last day

SELECT LAST\_DAY(SYSDATE) AS end\_of\_month FROM dual;

--months difference

SELECT MONTHS\_BETWEEN(SYSDATE, TO\_DATE('01-JAN-2025', 'DD-MON-YYYY')) AS months\_diff

FROM dual;

--least and greatest

SELECT LEAST(SYSDATE, TO\_DATE('31-DEC-2025', 'DD-MM-YYYY')) AS earliest,

GREATEST(SYSDATE, TO\_DATE('01-JAN-2025', 'DD-MM-YYYY')) AS latest

FROM dual;

--truncate

SELECT TRUNC(SYSDATE) AS today\_no\_time,

TRUNC(SYSDATE, 'MM') AS first\_day\_of\_month,

TRUNC(SYSDATE, 'YYYY') AS first\_day\_of\_year

FROM dual;

--ruund date

SELECT ROUND(SYSDATE, 'MM') AS rounded\_month,

ROUND(SYSDATE, 'YYYY') AS rounded\_year

FROM dual;

--To char

SELECT TO\_CHAR(SYSDATE, 'DD-Mon-YYYY HH:MI:SS AM') AS formatted

FROM dual;

--To date string

SELECT TO\_DATE('15-03-2025', 'DD-MM-YYYY') AS real\_date

FROM dual;

B) Program development using creation of stored functions, invoke functions in SQL Statements.

CREATE OR REPLACE FUNCTION get\_annual\_salary(p\_emp\_id IN Employees.emp\_id%TYPE)

RETURN NUMBER

IS

v\_salary Employees.salary%TYPE;

v\_annual\_salary NUMBER;

BEGIN

-- Get salary of employee

SELECT salary INTO v\_salary

FROM Employees

WHERE emp\_id = p\_emp\_id;

-- Calculate annual salary

v\_annual\_salary := v\_salary \* 12;

RETURN v\_annual\_salary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- If employee ID is invalid

WHEN OTHERS THEN

RETURN -1; -- Return -1 for unexpected errors

END;

/

--invoke the function

SELECT get\_annual\_salary(1) AS annual\_salary FROM dual;

--select statement

SELECT emp\_id, name, get\_annual\_salary(emp\_id) AS annual\_salary

FROM Employees;

--calculate bonus

SELECT name

FROM Employees

WHERE get\_annual\_salary(emp\_id) > 650000;